PCT/GB 99 / 02138

The Patent Office

INVESTOR IN PEOPLE

GB 99/02138

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

PRIORITY DOCUMENT

SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated 8th July, 1999

Patents Form 1/77

The Patent Office

08APR99 E438286-1 D00032
P01/7700 0.00 - 9907918.8

Patents Act 1977
(Rule 16)

-7 APR 1999

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference — PJF10072GB

2. Patent application number
(The Patent Office will fill in this part) — 9907918.8

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

PETER GRAHAM CRAVEN
6 Kings Road
Lancing
West Sussex
BN15 8EA
United Kingdom

Patents ADP number *(if you know it)*

If the applicant is a corporate body, give the country/state of its incorporation — United Kingdom

4. Title of the invention

LOSSLESS PACKING

5. Name of your agent *(if you have one)* — ELKINGTON AND FIFE

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

ELKINGTON AND FIFE
PROSPECT HOUSE
8 PEMBROKE ROAD
SEVENOAKS
KENT
TN13 1XR

Patents ADP number *(if you know it)* — 67004

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of Filing *(day/month/year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of Filing *(day/month/year)*

8. Is a statement of inventorship and of right to grant of a patent required in support of this request? *(Answer "Yes" if:*
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.

See note (d))

N

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	1
Description	48
Claim(s)	0
Abstract	0
Drawing(s)	0

10. If you are also filing any of the following, state how many against each item.

Priority documents	0
Translations of priority documents	0
Statement of inventorship and right to grant of a patent *(Patents Form 7/77)*	0
Request for preliminary examination and search *(Patents Form 9/77)*	0
Request for substantive examination *(Patents Form 10/77)*	0
Any other documents *(please specify)*	0

11. I/We request the grant of a patent on the basis of this application.

Signature

Date
7 April 1999

12. Name and daytime telephone number of person to contact in the United Kingdom

Mr Peter Finnie
0171 405 3505

Patents Form 1/77 Continued

3. Full name, address and postcode of the or of each applicant

MALCOLM JAMES LAW
14 Stonecroft Close
Hangleton
Hove
East Sussex
BN3 8BP
United Kingdom

J ROBERT STUART
21 Storeys Way
Cambridge
CB3 0DP
United Kingdom

Contents

1 Overview

Meridian Lossless Packing (MLP) has been designed to perform lossless compression of high quality audio data, including audio sampled at higher rates such as 96 and 192kHz.

It provides the following features:

- Good compression of both peak and average data rates
- Efficient use of both fixed-rate and variable-rate data-streams
- Automatic savings on bass-effects channels, without special flagging
- Automatic savings on signals sampled at 96 or 192kHz that do not use all of the available bandwidth
- Modest decoding requirements

The reduction of peak data rate is equivalent to reducing the wordwidth by 4 bits or more with 48kHz-sampled signals, or by 8 bits with 96kHz-sampled signals. Thus 24-bit 96kHz audio is effectively compressed to 16 bits, making it possible to record 4 channels of 24-bit 96kHz audio on a 6·144Mbit/s DVD stream, or 6 channels in a 9·6Mbit/s stream.

The specification is extremely flexible as regards multichannel operation. The number of channels can be very large, being limited by the available data rate, and a standard decoder that can handle only 6 channels may decode a stream containing a larger number of channels. There is also provision for low-cost and portable applications, whereby a 2-channel decoder with small MIPS and memory requirements can recover an $\{L_0, R_0\}$ mix from a multichannel track.

1.1 This document and other documents

Sections 1–5 of this document summarise the characteristics and performance of MLP. Sections 6–8 describe briefly the bitstream and the technical features of the compression system algorithm. Section 9 provides a detailed syntax for the compressed bitstream. The appendices A, B and C describe some of the additional (non-audio) data that MLP can convey, such as *channel meaning* information.

Section 10 refers to bibliographic material pertinent to MLP and its context.

A reference decoder for the MLP bitstream, written in *C*, is given in reference [5].

MLP has been designed for general use. Format-specific applications may be covered in supplements to this document.[1]

2 Available compression

At 44·1 or 48 kHz, the peak data rate can almost always be reduced by at least 4 bits/sample, i.e. 16-bit audio can be losslessly compressed to fit into a 12-bit channel.

At 96kHz, the peak data rate can similarly be reduced by 8 bits/sample, i.e. 24-bit audio can be compressed to 16 bits and 16-bit 96kHz audio can be losslessly compressed to fit into an 8-bit channel.

When making comparisons between systems, it is important to be consistent about whether peak or average rates are being compared. Descriptions of other compression systems often quote their average compression rates, which benefit from any extended quiet passages, and fail to mention the compression on peak passages, which will always be worse. Moreover, the extent to which it is worse may be greater than in MLP, in which peak compression has been given particular attention.

[1] In the case of DVD Audio, the specification document published by the DVD Forum describes how the parameters in the general version of MLP are specialised for use on DVD Audio.

In Table 1 we quote data rate savings for both peak and average passages. The peak rates are for 'difficult' signals, while the range of figures given for average data rates reflects the uncertainty introduced by the presence of quiet passages and other variables.

Sampling frequency kHz	Peak data rate saving bits/sample	Average data rate saving bits/sample
44·1	4	(5–11)
96	8	(9–13)
192	9	(9–14)

Table 1. Reduction in peak and average data rates using MLP in lossless mode. Figures are for difficult audio signals using 1998 encoding technology.

Elsewhere in this document we generally quote peak data rates, as the peak rate delivered by a stream (for example 6·144 or 9·6 Mbit/s) determines the maximum number of channels. If the stream is padded to a *fixed data rate*, the peak data rate from the encoder becomes the (constant) data rate of the stream.

3 Multichannel aspects

A 9·6Mbit/s stream can typically accommodate 6 channels of losslessly compressed 24-bit 96kHz audio. At 44.1kHz, this data rate could accommodate at least 10 channels of losslessly compressed 24-bit audio, or up to 24 channels in favourable cases with a 16-bit source. Signals of different wordwidth can be mixed on different channels, so a carrier could convey the standard '5·1' channels with high resolution, and additional channels with reduced resolution.[2]

We envisage three categories of decoder for the consumer market:

- 2-channel decoder
- Standard: able to decode 6 channels
- Extended: able to decode more than 6 channels

If a carrier contains more than 6 channels, the 'standard' decoder will decode the first 6.

The 2-channel decoder can recover an $\{L_0, R_0\}$ downmix from the multichannel mix. This facility has been incorporated in order to permit economical decoding for low-cost and portable players.

4 Application and structure of MLP

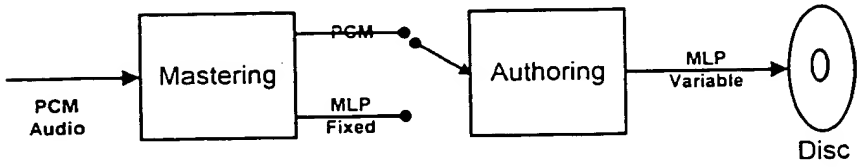
The MLP system provides a core compression method, which reduces the data size and/or data rate of an audio object. In terms of the encoder operation, this core may be embodied in a BIN or binary disk file[3] which can be decoded directly to recover the original audio.

MLP-compressed audio is normally then given a packetising layer in a manner that suits the target transport method. Obvious transport mechanisms include computer disk, DVD disc, SPDIF interface and Firewire interface. For each of these fixed-rate or variable-rate streams can be envisaged.

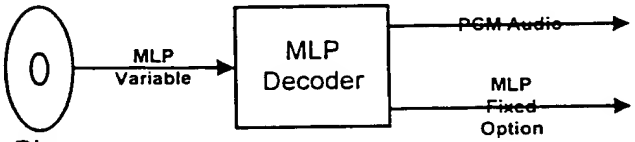
[2] Reduced resolution includes the possibility of reduced bandwidth. In this case the additional channels are transmitted at the same sampling frequency, but the fact that the bandwidth is reduced results in an automatic saving in data rate.

[3] Binary files have so far been generated by software encoders.

For these transport systems MLP has been structured so that the core coded audio can be packetised into fixed-rate or variable-rate streams *and* so that a re-packetiser can convert MLP-encoded audio between the transport variants *and/or* between fixed-rate and variable-rate streams without requiring an intermediate decode-encode process.[4]



Lossless encoder at mastering or authoring



Lossless decoder

Figure 0 How mixed-rate MLP streams can provide extra functionality and protection. The streams on disc can be either fixed or variable rate, but it is possible for a content provider to encode fixed-rate streams in an editing environment. Fixed-rate MLP can be re-packetised at authoring into a variable-rate stream without decoding, thus preserving the inherent protection and side information MLP offers to the audio. The decoder may optionally be configured to provide a fixed-rate MLP stream for transfer beyond the player (e.g. to a surround controller).

5 Decoder hardware

MLP is designed to allow low-cost decoding using either hardware or software.

The most demanding multiplication is that of a 16-bit operand by a 24-bit operand. This keeps down the size of the multiplier in a hardware implementation. For a software implementation, a 24-bit DSP will give the lowest number of operations, but very reasonable performance will also be obtained on a 16-bit DSP with a 32-bit accumulator.[5]

On a Motorola DSP56303, 6 channels of 96kHz audio may be decoded in less than 40 MIPS.

2-channel replay (including $\{L_0, R_0\}$ replay of a 5·1-channel track) can be done in approximately 15MIPS at 96kHz, or 27MIPS at 192kHz.

The standard decoder requires 90,000 bytes of memory in order to implement the FIFO latency buffer mentioned in section 6.6. This memory is accessed infrequently and slow external RAM is very adequate, but it is possible to implement the standard decoder on a DSP such as the Motorola DSP56309 without using external RAM.

[4] This aspect of MLP packetising is the subject of a patent application.

[5] On a 16-bit DSP, 24-bit signals require some double-precision working, of course, but expensive full double-precision multiplications are not necessary.

The 2-channel decoder can use less memory: approximately 3k bytes.[6] Thus 2-channel decoding is possible using only internal memory on popular DSPs, and a single-chip implementation is possible on inexpensive DSP parts.

6 Overall view

6.1 Bitstream

The MLP bitstream is a flexible format for describing multichannel audio. To decode a large number of channels at a high sampling rate will, however, always be a computationally demanding task. Consequently MLP has been defined in a hierarchical manner so that decoders of lesser capability can easily extract the audio signals they require, skipping over parts that are intended for more advanced decoders.

The MLP bitstream carries a number of *substreams* containing the audio data. The number of substreams will depend on the application. For example, 2-channel decoders only need to decode substream 0; standard multichannel decoders must decode substream 0, substream 1 or both.

In general additional substreams may be provided within the MLP stream for use by more advanced decoders.

6.2 Encoder and decoder

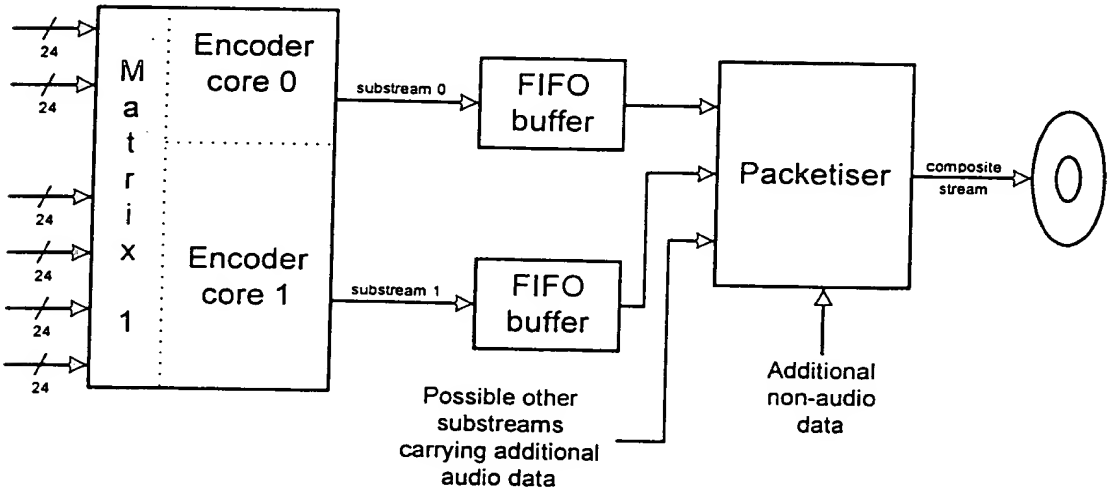


Figure 1. Overview of encoder structure

As shown in figure 1, the encoder takes the input channels and divides them (possibly after matrixing) into groups appropriate to the various classes of decoder. Each group is then processed by an *encoder core* to produce a *substream* of variable-rate compressed data.

For example, a normal 5·1 channel disc will have 6 channels which can be decoded by a standard decoder. These would be matrixed and divided into groups of 2 and 4 channels, the matrix being chosen so that the 2-channel signal is an acceptable mix for the 2-channel listener. The two groups are then each encoded by separate encoder cores to produce substreams 0 and 1.

The encoder passes each substream through a FIFO buffer to the *packetiser*, which interleaves the substreams to produce the composite bitstream, consisting of a regular stream

[6] However, the constraint of common substream delay in some applications will increase this requirement to 30,000 bytes of memory for a 2-channel DVD decoder.

of *packets* or *access units*. If a fixed-rate stream is required, the packetiser pads the variable-rate stream to the desired fixed rate. Optionally, additional data may be added at this point, and these data can occupy the space that would otherwise be wasted.

In the generic MLP decoder (see figure 2), the *depacketiser* receives the packets or access units and retrieves the substreams, which it places in one or more FIFO buffers. It may optionally recover any additional data at this point. The data in each FIFO buffer will be a pure substream with all packet-level information removed.[7]

After buffering, each substream is passed to a *decoder core*. In the simple case where a substream contains the data for a completely independent group of channels, the decoder core recovers these channels. Figure 2 illustrates the more advanced case where the matrixing in the encoder has spread information across substream boundaries (see section 6.4). A unique feature of MLP is *lossless matrixing*, which allows exact recovery of the original signal, without the rounding errors expected from the standard use of matrices.

6.3 Input and output formats: absence of flagging

The audio inputs are presented to the encoder through 24-bit wide data-paths.

However, an MLP compressed signal will take up only as much data rate as the information it contains. If the encoder is presented with a signal containing less information then compression will automatically achieve a reduction in required data rate. Common situations resulting in a great reduction in data rate include:

- Silence (digital black), which will automatically be transmitted at virtually zero data rate.[8]
- A signal that exercises fewer than 24 bits, either through being at less than peak level, or through being quantised to fewer than 24 bits.
- A signal that occupies less than the full bandwidth permitted by the sampling rate (a bass effects channel is an extreme example).
- Channels that show a strong linear dependency. For example, it is no more expensive to transmit a mono signal as two identical channels than as one channel. Similarly, two identical surround channels will occupy the same data rate as one.

All of these situations will automatically be recognised by the encoder, and the appropriate economies of data rate will ensue. Flagging of these situations is totally unnecessary as far as the compression and decompression processes are concerned.

Of course, for subsequent processing it can be extremely helpful to have these special situations identified. This is done by means of the *channel meaning* information (appendix A) that is presented to the packetiser as part of the 'additional data' in figure 1. However, we emphasise that these additional data are stripped by the depacketiser of figure 2 and play no part in the lossless decoding of the audio signals.

We emphasise that *low-frequency-effects channels* do not need to be identified as such. They are presented to MLP like any other signal, and benefit automatically from a low data rate on account of the lack of high-frequency information.

---

[7] Alternative architectures are possible if all relevant substreams have identical FIFO delays, as is the case with the standard DVD decoders. A single FIFO buffer can be placed before the depacketiser, or alternatively a multichannel FIFO can be placed at the output. The memory requirement for the first of these options will be about 100kbytes. For the second it will be enough to hold 75ms of decoded audio.

[8] The MLP bitstream has a certain overhead data rate that is controlled by encoder settings. For 6 channels at 96kHz this rate is approximately 120kb/s.

6.4 Matrixing

As shown in figures 1 and 2, the encoder and decoder cores each incorporate a matrix (in fact a *lossless matrix*: section 7.2). The matrix allows linear dependencies within the group of channels to be exploited in order to reduce the data rate.[9]

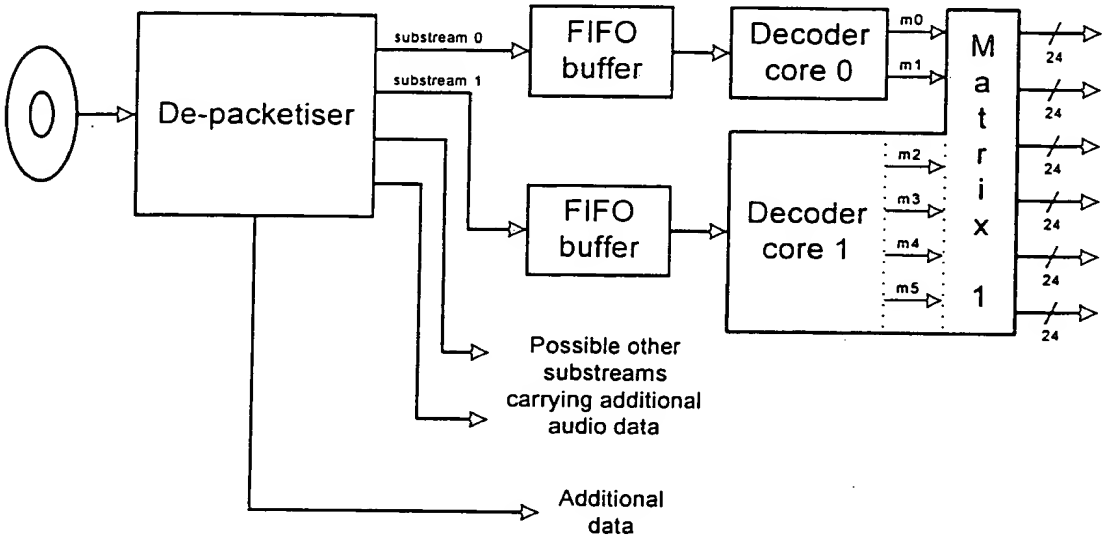


Figure 2. Overview of a decoder when decoding substreams 0 and 1

When several substreams carry the data for a group of channels, the last substream carries the necessary matrix coefficients for the whole group. Thus, in the example shown in figure 2, substream 1 carries the data for four channels, plus the matrix coefficients for six channels. Decoder 1 partially decodes the four channels of substream 1, then takes in two partially decoded channels from decoder 0, and all six channels participate in the final matrixing.

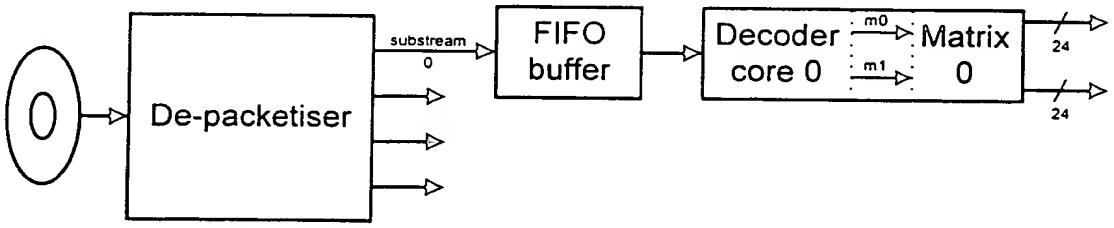


Figure 3. Overview of decoder decoding substream 0 only

Substream 0 also contains matrixing information, but this is used only if substream 0 is decoded in isolation (see figure 3). It follows that the two signals that result from decoding substream 0 alone need not be identical to the first two signals that result from decoding substreams 0 and 1. This is the key to the economical decoding of an $\{L_0, R_0\}$ downmix, as described in the next section.

6.5 Economical decoding of $\{L_0, R_0\}$

Many multichannel applications require that the 2-channel listener be catered for. One possibility is to record a separate 2-channel mix, but this is wasteful of storage space (or playing time on a disc). Another option is downmixing whereby the $\{L_0, R_0\}$ signals are

[9] For example, if two channels are very nearly the same as each other, the matrix may replace the second channel by the difference between the two, which will encode to a lower data rate. The encoder may optimise the general case by calculating the eigenvectors of a suitable matrix, as explained in reference [4].

derived at replay from the multichannel signal. However, this would require 2-channel playback hardware to provide enough MIPS to decode the full multichannel signal.

MLP specifically solves this problem by using the fact that it is possible to recover (for example) a 6-channel original signal from a 2-channel mixdown plus four other signals.[10]

Thus, we transmit L_0 and R_0 (or rather, two signals m_0 and m_1 that can be matrixed to L_0 and R_0) in substream 0, and the four other signals in substream 1. The 2-channel decoder is then able to recover the $\{L_0, R_0\}$ signals with minimal effort (see figure 3), while a full decoder retrieves the original (in this example) 6 channels.

The principle of transmitting L_0 and R_0 plus supplementary signals is used in other compression systems, for example MPEG 2 AAC [3]. The difference in MLP is that *lossless matrixing* is used so that the channels can be recovered with bit-for-bit accuracy (see reference [4]). Moreover, MLP also defines standardised dither channels, and the mixdown for L_0 and R_0 can be dithered to audiophile standards without impeding exact regeneration of the multiple channels.[11]

The coefficients for the desired $\{L_0, R_0\}$ mix are notified to the encoder so that the lossless Matrix 1 of figure 1 can be configured suitably.[12] The total effect of the encoding of figure 1 and the decoding of figure 3 is to furnish $\{L_0, R_0\}$ as a properly dithered mix from the original six input signals, using the specified coefficients.

6.6 Buffering, latency and cueing

Each encoder core produces a variable-rate substream, the data rate being greatest during peaks of high treble energy. The FIFO buffers in figure 1 are crucial in reducing the peak data rate on the disc. These FIFO buffers in the encoder fill during passages of peak data rate from the encoder cores, and empty when the data rates from the encoder cores are lower than the maximum data rate of the transmission medium or carrier.

Correspondingly, the FIFO buffers in the decoder (figure 2) are filled during passages of lower data rate, and empty during passages of peak rate, thus allowing peak data rates higher than the transmission maximum to be delivered to the decoder cores.

---

[10] In mathematical terms, the original 6 signals may be considered to span a 6-dimensional vector space. The 2-channel mixdown spans a 2-dimensional subspace and the 'other' signals can in principle be any other 4 signals such that all 6 are linearly independent.

[11] Since the mixdown function is in the encoding phase, the content provider can also listen to the mixdown and approve it. This mixdown is then delivered losslessly by MLP. Mixdown coefficients may be changed (with no penalty in data rate) at every restart in the stream [9.9.2].

[12] This is a constraint on Matrix 1, whose original purpose was to minimise the transmitted data rate. The constraint will increase the data rate, but the increase will be small (usually less than one bit per multichannel sample).

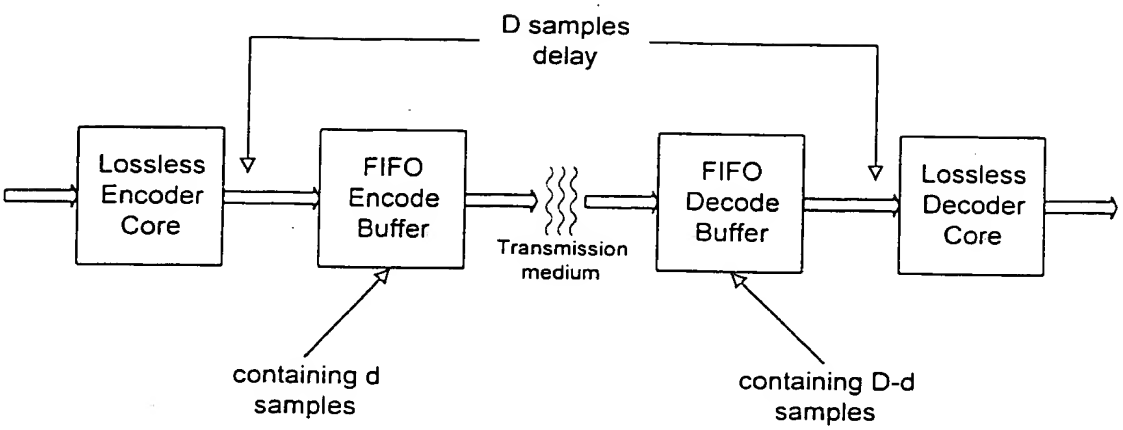


Figure 4. Use of FIFO buffers to smooth the data rate for use with transmission media with limited peak information capacity

Buffering introduces delay, and the delay is variable as the buffers fill and empty. Figure 4 highlights the delay aspects involved in the encode-decode process: it is clear that when a FIFO buffer in the encoder fills, the corresponding FIFO buffer in the decoder must empty, so that the total delay D is constant.

On typical audio signals the data rate fluctuates substantially over a period of a few tens of milliseconds, and FIFO buffering with a total delay D of order 50–100ms generally reduces the peak data rate by about 2 bits per sample. This gives an advantage of nearly 1Mbits/s for 5 channels sampled at 96kHz.

When the transmission does not take place in real-time, as with disc recording, the total delay D in the encoding and decoding is not a relevant consideration. Operationally, the important issue is the *decode latency*, which directly affects the cueing time experienced by the user.[13] A major component of this is the *buffer latency*, which is simply the delay through the decoder's FIFO buffer.

The maximum buffer latency in the standard application is 75ms, but for the vast majority of the time the latency will be approximately 1ms. The filling and emptying of the decoder's FIFO buffer is under the control of the encoder, which arranges that the decoder's buffer is empty for most of the time (giving very low buffer latency), but fills just before passages that result in the highest rate of compressed data, for example one containing a cymbal crash. Thus it is only immediately prior to such a peak event that the buffer latency will be near its maximum value.

The standard decoder requires 90,000 bytes of buffer memory, but a 2-channel decoder (section 3) can use less than 3kbytes total. This is because each substream is separately buffered (figure 1) and buffering can be removed from an $\{L_0, R_0\}$ substream with no impact on data rate.[14]

Taking into account the time taken to find the various headers, the total decode latency at 96kHz is between 2 and 10ms during normal passages, with a worst case of 105 ms immediately before a peak.

---

[13] The decode latency is defined as the time between the decoder first receiving the compressed data stream and being able to produce decoded samples.

[14] This can be understood as follows. The effect of the buffering is to move the data on the disc away from the places where the data rate would otherwise exceed the capability of the disc. However, an $\{L_0, R_0\}$ substream would never, on its own, exceed the data-rate capability of the disc. Hence the $\{L_0, R_0\}$ substream can be recorded on the disc with virtually no buffering, provided the other substream(s) are given extra buffering so that their data are moved well away from the peak places.

7 Encoder and decoder cores

Reference [1] contains an introduction to some of the principles used in MLP.

As noted previously, each encoder core and decoder core processes the *n* channels conveyed by one substream.

The signal-flow diagrams for the encoder core and decoder core respectively are shown in figures 5 and 6. These refer to the sample-by-sample processing and are mirror images of each other.

7.1 Encoder core

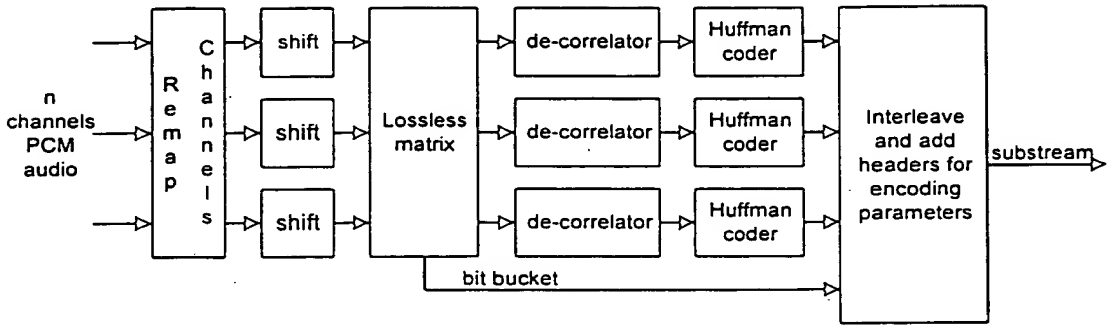


Figure 5. Encoder core

First the input channels are remapped to more suitable channel numbers for the codec's internal operation.[15] Then an *input shift* is applied to each channel, after which the signals are passed to a *lossless matrix* (section 7.3). Each matrixed channel is then passed through a de-correlator[16] (section 7.4) and the resulting samples are Huffman coded (section 7.5).

In passing through this processing the signal is re-quantised many times. All these quantisations are done in a precisely defined manner, which is the key to ensuring a lossless decode.

The Huffman-coded samples from the *n* channels are now interleaved[17] to produce the composite Huffman-coded substream. This substream is then organised into blocks (not shown), and the various parameters used in the encoding are inserted into the block headers. The first block header after a restart header must specify all the encoding parameters; the second and subsequent block headers need include only the parameters that have changed.

The choice of encoding parameters is crucial to good compression performance, and is the most difficult aspect of encoder design, making encoders considerably more complex than decoders: we do not consider it further in this document.

---

[15] For example, a 3-channel signal containing Lf, Rf and C might be presented on channels 0, 1 and 4. The codec would find it more convenient to re-map to channels 0, 1 and 2. Re-mapping is also sometimes needed to ensure correct operation of the $\{L_0, R_0\}$ feature.

[16] This is a temporal de-correlator (sometimes called a 'predictor') and is distinct from the inter-channel decorrelation provided by the lossless matrix.

[17] One *bit-bucket* (section 7.3) per sample is also interleaved.

7.2 Decoder core

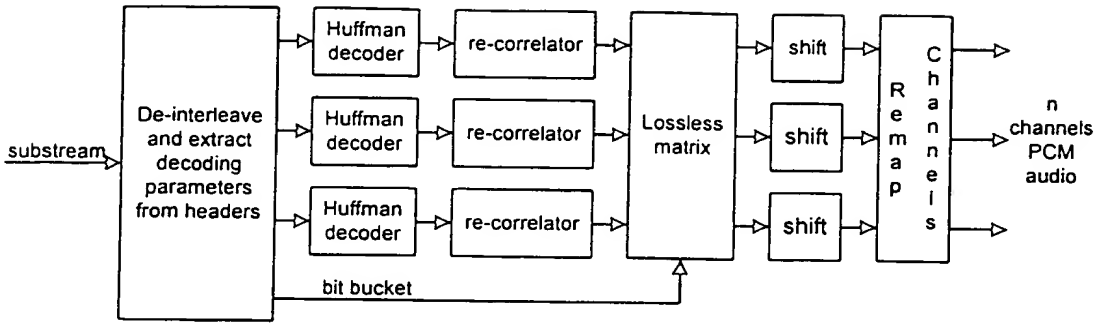


Figure 6. Decoder core

The decoder core inverts the processes performed by the encoder core, in reverse order.

The incoming substream is first parsed to extract the parameters that control the decoding. This leaves an interleaved sequence of Huffman-coded samples, which are de-interleaved and decoded. Each channel is then passed through a re-correlator, the inverse of the de-correlator. A lossless matrix inverts the effect of the encoder's lossless matrix, and then an *output shift*, the converse of the input shift, reconstitutes the original data. The decoded channels are then remapped to the channel numbers that were originally presented to the encoder.

Once again all signal quantisations are performed in a simple but precisely defined manner to ensure that the final output from the decoder is bit for bit identical to the input to the encoder.

7.3 Lossless matrix

Matrixing is used to minimise inter-channel dependency, and hence the total transmitted data rate. For example, if two channels are very similar it is more efficient to transmit one of them and the difference between the two.

It is not adequate for the decoder simply to multiply by the inverse of the encoder's matrix, as the rounding errors involved in the matrix multiplications will result in lossy reconstruction of the original. This problem is overcome using *lossless matrixing* [4], in which the encode matrix includes carefully placed quantisers which ensure that the rounding errors are precisely known and can be cancelled using similar quantisers in the decoder.

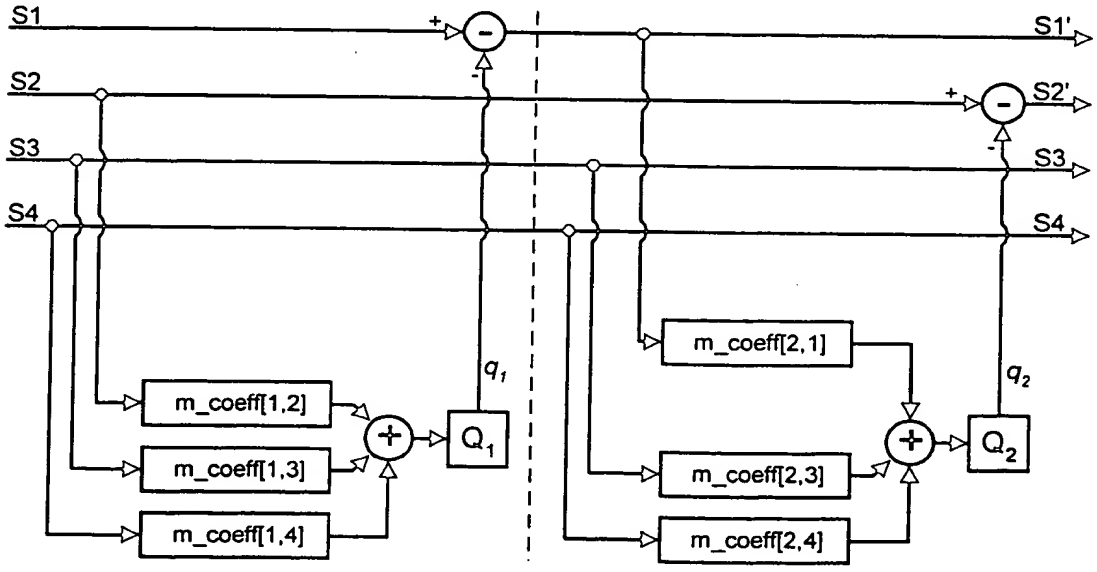


Figure 7a. Example of lossless matrix encoding, comprising two cascaded primitive matrices

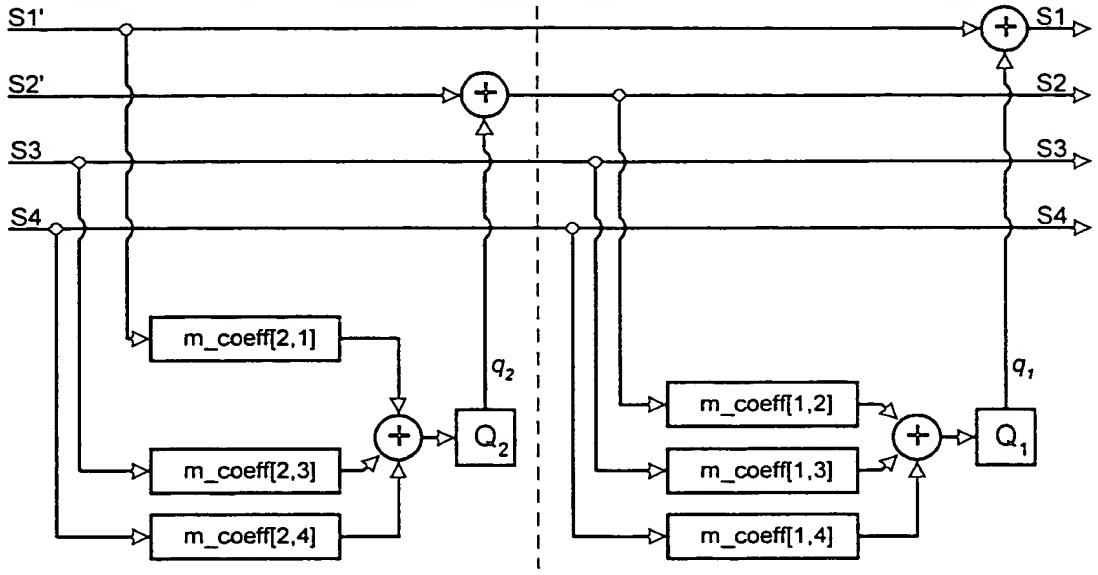


Figure 7b. Lossless matrix decoding

Each lossless matrix is a cascade of *primitive matrices*:[18] each primitive matrix modifies just one channel. The principle is shown in figure 7, where figure 7a shows a cascade of two primitive matrices used to modify channels S1 and S2 in the encoder, while figure 7b shows the converse decoder architecture that restores the original signals.[19]

This lossless matrix may operate over more channels than those produced by the core decoder, as illustrated in figure 2. As shown, the matrixing applied by the decoder core 1 may operate both on its output and on the (unmatrixed) output of decoder core 0.[20] Each substream also defines two standardised *dither channels*, which may be mixed in by primitive matrices.[21]

MLP carries 24-bit audio while requiring only a 16×24-bit multiply. Thus overload of intermediate results in the signal processing is a potential problem. To avoid this problem each of the primitive matrices in the encoder optionally removes one bit from the modified signal and transfers it to a *bit-bucket*. The bit-bucket resulting from all the primitive matrices on a sample is then transmitted along with their Huffman-coded values. In the decoder, the bit from the bit-bucket is inserted into the signal at the appropriate primitive matrix to restore lossless operation.

---

[18] On DVD, an MLP substream intended for a standard decoder may require up to six primitive matrices, each having an architecture derived from that in figure 7b.

[19] To verify bit-for-bit reconstruction, observe that the quantiser Q_2 in figure 7b is fed with the same signal as the quantiser Q_2 in figure 7a. They therefore produce the same output q_2. In figure 7a the signal S2′ is formed as $S2' = S2 - q_2$, while figure 7b performs the restoration $S2 = S2' + q_2$. With $S2$ thus restored, quantiser Q'_1 is fed with the same signal as quantiser Q_1, and signal $S1$ is restored in the same way as $S2$.

The quantisers Q_1 and Q_2 are needed in order to prevent the wordlength of the modified signals $S1'$ and $S2'$ from exceeding that of the input signals $S1$ and $S2$, so that the information content and hence the transmitted data rate is not increased.

[20] This allows any linearly independent mixdown of the channels to be carried in a separate substream with only a mild penalty in data rate. See section 6.5.

[21] This allows the mixdown to be dithered to audiophile standards whilst still allowing lossless reconstruction of the original multichannel signal.

7.4 De-correlator and re-correlator

The de-correlator used in the encoder is discussed extensively in references [2] and [4]. Its purpose is to 'whiten' the signal spectrum, i.e. to remove correlations with previous samples and thereby reduce the sample amplitudes as much as possible.

De-correlation is performed by passing the signal through an 8 coefficient IIR (Infinite Impulse Response) filter. This filter is calculated by the encoder so that its transfer function is as close as possible to the inverse of the signal spectrum. In the decoder, the converse operation of *re-correlation* is performed using the inverse IIR filter. As with matrixing, it is necessary to make special rounding provisions if the reconstruction is to be lossless.

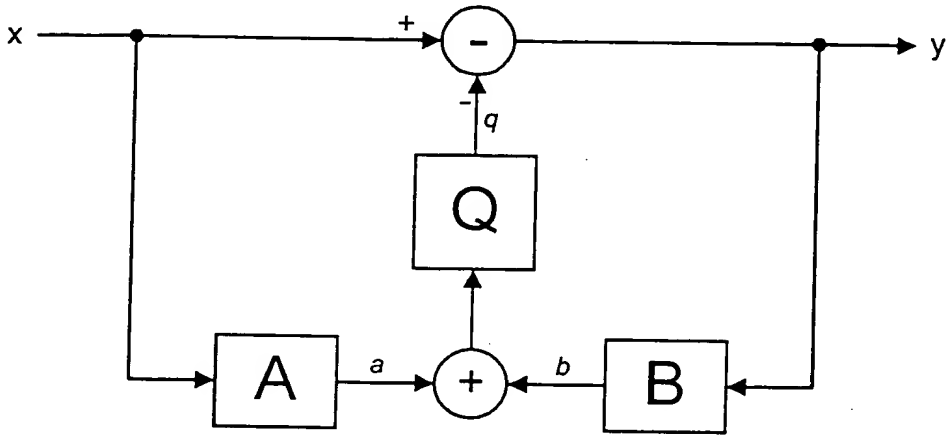


Figure 8a. Lossless de-correlator (encoder)

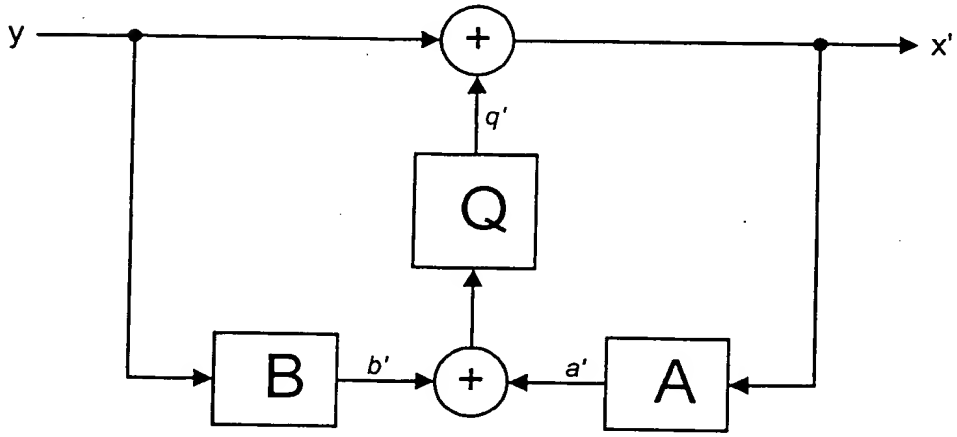


Figure 8b. Lossless de-correlator (decoder)

Figures 8a and 8b show the simple lossless IIR filter architectures from which the filters used in MLP are derived. In figure 8a, FIR filter A implements the numerator of the encoder's IIR filter, while FIR filter B implements the denominator. The decoder (figure 8b) contains identical filters A′ and B′.

It can be shown[22] that the figure 8b decoder achieves lossless reconstruction provided that the filters A and B in the decoder are initialised to the same internal state as filters A and B in the

[22] Figure 8a implements $y = x-q$, while figure 8b implements $x = y+q'$. FIR filters A and B have no straight-through path (no terms in z^0), so if the decoder filters A and B are initialised to the same states as the encoder filters A and B, then $a' = a$ and $b' = b$ on the first sample period. Therefore $q' = q$, and hence also $x' = x$ on the first sample period. It follows that $q' = q$ on the second sample period, and (by induction) that equality will be maintained on subsequent samples.

encoder. The encoder has the ability transmit the necessary state information to the decoder in order to ensure that the initial states are identical.

7.5 Huffman coding

Huffman coding is a widely used technique for saving data rate when not all possible values are equally likely. MLP uses 4 different Huffman tables, including the well known Rice code, to cater for differing signal statistics. These tables are all designed to scale with signal level and are simple to decode algorithmically (not using tables), though it will often be more efficient to use tables in software decoders.

As the length of a Huffman-coded sample is not known until it is decoded and the Huffman-coded samples are interleaved together on a sample-by-sample basis, the Huffman decoder must combine the operations of de-interleaving and decoding.

8 Bitstream organisation

8.1 External and internal structure; forms A and B

The MLP stream has two levels of organisation, external and internal. The external organisation is designed for easy handling by an external system, while the internal structure is designed for efficient coding of the audio.

The internal structure is based on *blocks* and *restart headers* (section 8.5). The external structure exists in more than one form. We describe two forms in this document: form A, based on *packets* and *subpackets* (section 8.3), and form B, based on *access units* and *MLP Syncs* (section 8.4). Encoders and decoders using form A have been produced for use with IEC958. Form B is used for DVD Audio applications

The relationship of the external to the internal structure is analogous to the construction of a document such as this one, which has an external structure consisting of pages (of interest to the bookbinder) and an internal structure consisting of sections and subsections (of more interest to the reader).

8.2 Fixed and variable-rate streams

The data rate from the core encoder is inherently variable. This variability can be handled in several different ways.

In a form A fixed-rate stream, the packets can be padded to a constant length, and can occur at fixed intervals, such as one every 1536 samples. This provides an extremely simple external interface, similar to that of fixed-rate lossy compression systems.

In a form A variable-rate stream, one option is simply to omit the padding from the corresponding fixed-rate stream, so that the packets again arrive at a constant rate but are of variable size. The notion of a corresponding fixed-rate stream has two advantages:

The quantiser Q in figure 8a ensures that the wordlength does not increase indefinitely as the signal circulates round filter B, which has fractional coefficients. This is the key to portability across platforms.

The topology used in MLP incorporates further modifications to reduce the noise build-up due to the recirculation round filter B.

- It is easy to write transcoders from the variable-rate format (which might be used for hard disc storage) to the fixed-rate format (for transmission over a serial link) and back again.
- The data rate of the variable-rate stream is never greater than that of the padded fixed-rate stream. Although variable, the data rate is therefore 'capped'.

Like the form A stream, the form B stream is flexible, and it can look very different depending on how its parameters have been configured. When the form B stream is configured for DVD Audio it is of variable rate, and each access unit contains the encoded data for a constant-length segment of audio. Consequently the access units are of variable size, and they arrive at non-constant intervals. A fixed-rate stream derived from this variable-rate stream will have access units arriving at the same non-constant intervals, each access unit being padded to fill the space between consecutive access units.

8.3 Form A: Packets and Subpackets

A *packet* consists of a *packet header* and one or more *subpackets*.

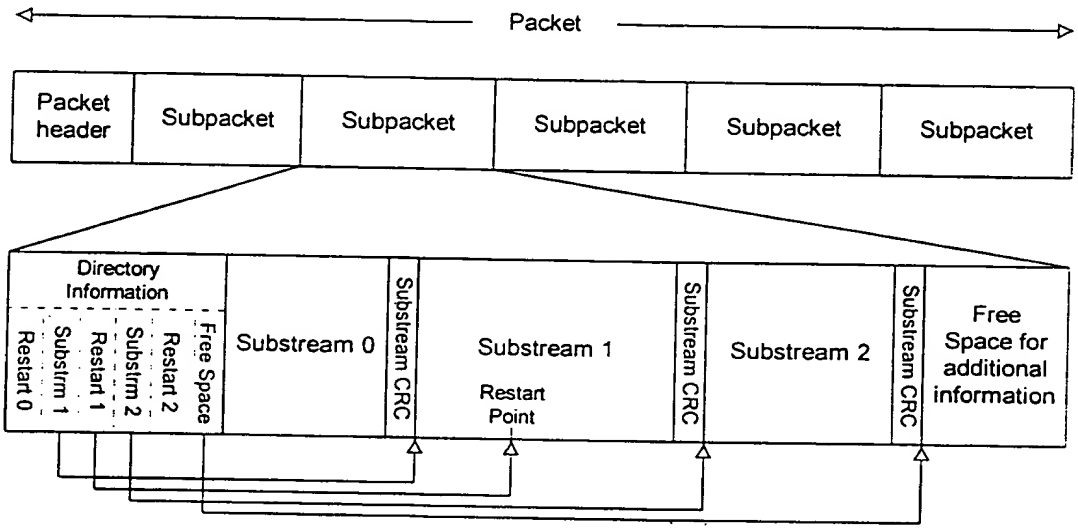


Figure 9a. A *packet*, containing 5 subpackets and 3 substreams. Substream 1 has a restart point in the illustrated subpacket.

The packet is designed to be compatible with an IEC958 *burst*. Thus, the *packet header* starts with the four 16-bit words P_a, P_b, P_c, P_d defined by IEC958, and an MLP form A stream of sufficiently low data rate will be compatible with the IEC958 standard. P_a and P_b are sync words, included so that the packet can be identified at any arbitrary position in a continuous bitstream.

The packet header is described in more detail in section 9.2. It contains items needed by the transport layer, such as packet length, and also *channel meaning* information (appendix A). However, all information needed by the core lossless decoder is contained within the substreams, not the packet header.

If there is more than one substream (section 6.1), the data from the various substreams are interleaved. Figure 9a illustrates that the packets may be broken into subpackets: this is done so that the substreams can be interleaved with finer granularity. Each subpacket starts with pointers so that a decoder can easily locate the substream(s) it wishes to decode.

Any padding required to pad the stream to a fixed data rate is placed at the end of each subpacket, as shown in figure 9a. This space may either be unused, or be filled with additional data.

8.4 Form B: Access Units and MLP Syncs

An *access unit* consists of an *MLP Sync* followed by the data for one or more substreams.

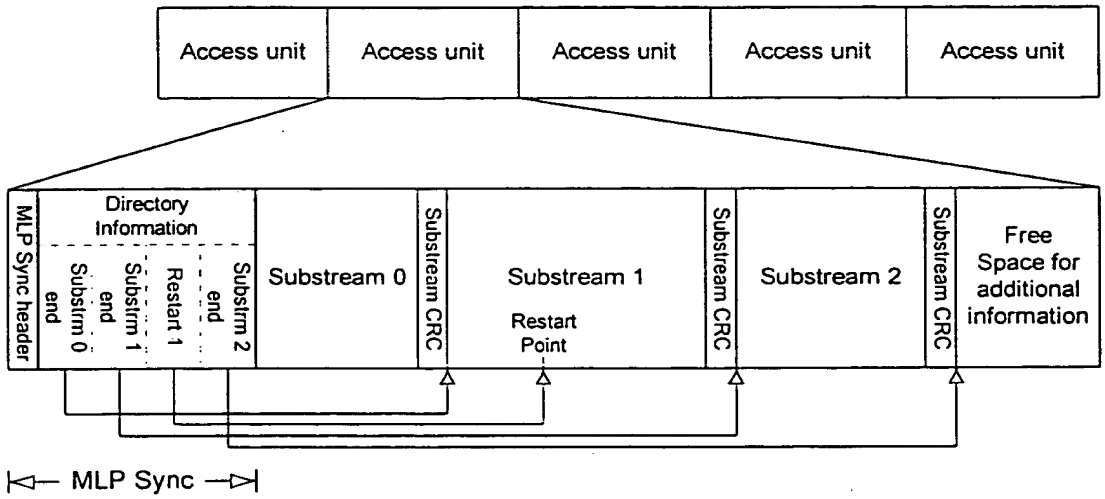


Figure 9b. A section of an MLP stream. The second access unit is expanded to show its internal structure, including the data for 3 substreams. Substream 1 has a restart header in the portion shown. The directory information in the MLP Sync points to this.

There are two types of MLP Sync: the *major sync* and the *minor sync*. (These correspond respectively to the packet headers and subpacket headers of the form A syntax.) Each sync contains a header and some directory information, as shown in figure 9b. The major sync has an expanded header containing all the information required to start full decoding of the stream (and other useful information, as with the packet header (section 8.3), whereas a minor sync's header consists of no more than size and time stamps. To minimise data rate overheads, most access units are introduced by minor syncs, major syncs occurring typically once per 8 access units.

Any padding required to pad the stream to a fixed data rate is placed at the end of each access unit.

8.5 Internal organisation: Blocks and Restart Blocks

The internal structure of each substream consists of *blocks* and *restart blocks*, each of the latter containing a *restart header* as well as a *block header*.

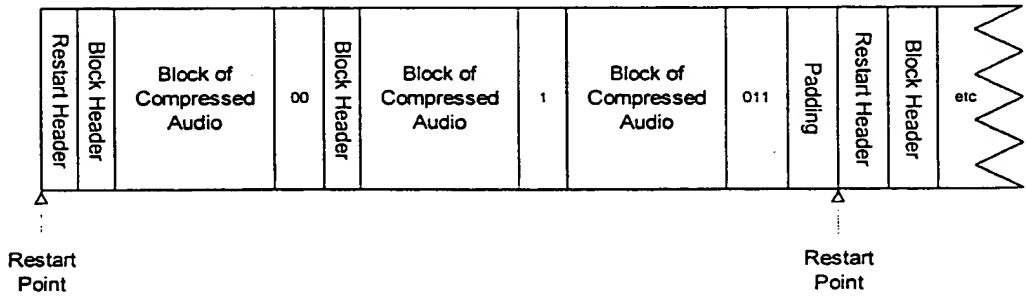


Figure 10. A section of a form B substream, showing that blocks of compressed audio may be juxtaposed without a block header, or with an intervening block header, or with padding and a restart header. These possibilities are signalled by the bit patterns '1', '00' and '011' respectively. The padding is to ensure that restart points occur on 16-bit boundaries.

Blocks can be of varying size, but will always contain a complete number of samples of the channels conveyed in the substream; restart points will be present at some of the block

boundaries.[23] As their name implies, restart points are the points at which decoding can be started, or restarted after an error. The *restart header* includes relevant initialisation information for the decoder.

Blocks may also begin with *block headers*, which permit selective updating of the decoding parameters so that the encoder can optimise its compression characteristics in response to changes in signal statistics.

Decoding of the audio data within a block can start as soon as the block header has been received; it is not necessary that the complete block should be received, as it is in FFT-based algorithms.

Some encoders will generate a regular sequence of restart points and blocks. However, the intervals are specified within the bitstream and can be altered midstream in order to allow flexibility in the encoding. For example, an encoder may choose to change parameters rapidly when the source material contains transients, in which case short blocks will be generated. (Care has been taken to minimise the block header overheads to allow this. In fact, the overhead need be only one bit if no parameters have changed.)

8.6 Relationship of external to internal organisation

In a form A stream, each substream (figure 10) is chopped at 16-bit boundaries to produce a segment that can be inserted into a subpacket (figure 9a). The boundary is chosen independently of the block and restart structure of the substream.

In terms of the previous analogy with the pages and sections of a document, this is equivalent to regarding the text of the document as a sequence of lines that is broken arbitrarily at page boundaries.

In MLP on DVD Audio each access unit consists of a complete number of blocks. Form B syntax contains a minor change (in *block()*, section 9.6) that supports this by allowing padding to the next 16-bit boundary before a block that is not necessarily a restart block.

8.7 Effect of the FIFO buffer on bitstream relationships

The effect of FIFO buffering will now be explained with regard to the form A and the form B stream.

8.7.1 FIFO buffering in the form A stream

Figure 11 shows how, as a consequence of buffering, the audio data are distributed non-uniformly in a form A fixed-rate stream.

[23] Typically blocks might be 40 to 160 samples long, with restart points inserted at intervals of 12ms. These figures are, however, entirely the encoder's choice, and should not be expected to be constant.

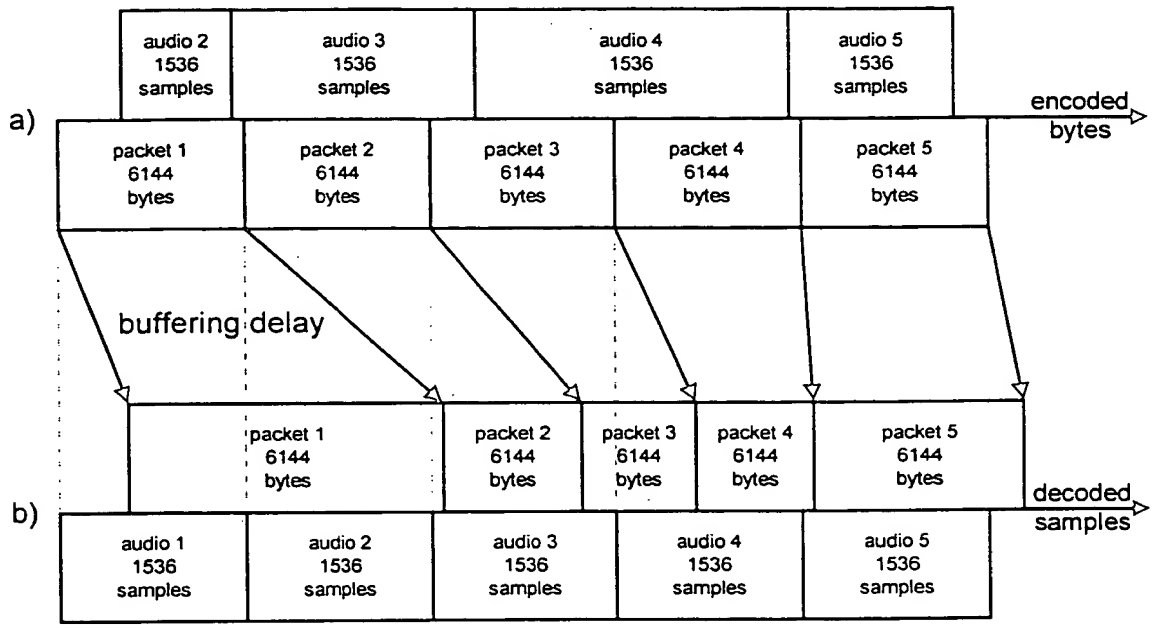


Figure 11. Relationship between packets in a form A stream and the audio data. See text.

In the upper part of figure 11 the horizontal axis represents encoded data, or, equivalently, a time axis corresponding to the time when the data are read from a fixed-rate transmission medium into the FIFO buffer. For the purposes of illustration the audio has been arbitrarily broken into segments of length 1536 samples (equivalent to the length of time taken to transmit 6144 bytes). However, because the data rate from the core encoder is variable, when the data are buffered to produce a stream whose rate is more nearly constant the audio segments will become stretched in time if the data rate is high (for example, in the 'audio 4' segment), and squashed when the data rate is lower.

In the lower part of the figure, the same data are plotted against a time axis corresponding to the time at which the data are read from the FIFO buffer into the decoder core. The buffer empties rapidly during passages of high data rate, so the decoder core receives nearly two packets' worth of data during the period labelled 'audio 4'. Conversely, the buffer fills during periods of low data rate, for example 'audio 1' and 'audio 2'.

Thus, data are fed to the decoder core at a variable rate, but the audio segments are played out at a constant rate.

The time difference between the head and the tail of the diagonal arrows represents the delay through the FIFO buffer. For example, at the end of packet 4 the arrow is nearly vertical and there is virtually no delay (the buffer is nearly empty).

Figure 11 illustrates that, while the internal timing relationships may be somewhat involved, the external behaviour of the decoder is very simple. In the first time slot packet 1 is read and audio segment 1 is decoded, and the other packets continue similarly. The decoder's external behaviour is as if packet 1 actually *contained* the data for audio segment 1, though the internal reality is very different.

8.7.2 FIFO buffering in the form B stream

FIFO buffering in the form B stream is illustrated here for DVD Audio.

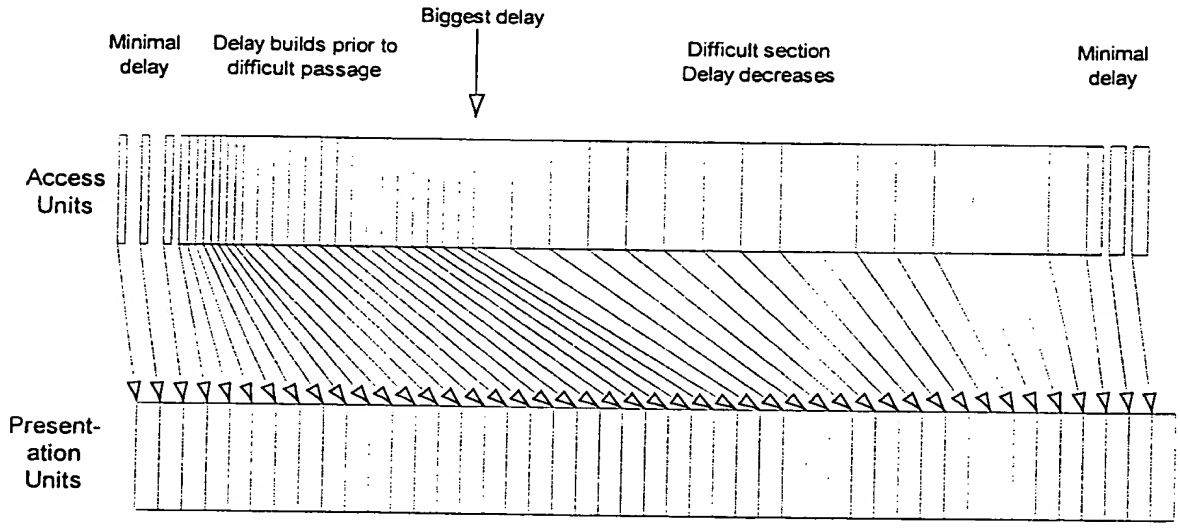


Figure 12. FIFO delay between receipt of access units and output of presentation units in DVD Audio. The deviation of the arrow from vertical represents the delay in the FIFO.

In this case (figure 12) the access units each decode to one presentation unit. At the beginning and end of the section, the audio is easy to compress and the access units are small enough to be read from the disc during one presentation unit. In this case the delay through the FIFO is just one presentation unit.

In the difficult section, the access units must be spaced out in order not to violate any constraint on peak data rate. As an access unit cannot arrive after its presentation unit is delivered, the access units for the earlier parts of the difficult section must be advanced in time. Thus the FIFO delay is maximal at the beginning of a section in which all the access units are each too big to be delivered during one presentation unit.

8.8 Decoder timing and FIFO management

The precise timing of the input to and output from the FIFOs in the decoder is derived from the *input_timing* field of the access unit or packet.[24] For ease of explanation, the fixed-rate decoder will be considered first, followed by a demand fed variable-rate decoder and finally an MPEG-compliant one. All these decoders are models. Practical decoders may be constructed differently.

8.8.1 Fixed-rate decoder

A fixed-rate encoded stream can be used to provide a timing reference for the decoder. The data rate of the input stream is assumed to bear a rational relationship to the sampling frequency, and the starts of packets or access units are always separated by an integer number of samples.

Each access unit or packet contains an *input_timing* field, and the decoder's sample clock is set to *input_timing* at the instant of arrival of the start of the access unit or packet.

[24] This is named *sample_no* in the packet syntax (section 9.3.1); '*input_timing*' in this discussion should be taken as including a packet *sample_no* in the case of a form A stream.

As figure 2 shows, the decoder consists of a depacketiser followed by a FIFO buffer and a core decoder for each substream. The depacketiser parses the outer syntax of the stream and feeds the data for each substream to the appropriate FIFO. However, the core decoder cannot decode a substream until it has found a restart header. Consequently the decoder waits until it encounters a restart point before feeding the substream to the FIFO.

The restart point contains an *output_timing* field. When the decoder's sample clock is equal to *output_timing*, it is time to decode and output the first sample.

A certain number of bits are fed to the decoder on each sample period, and the depacketiser transfers the appropriate bits from the input to each FIFO. The core decoder then decodes one sample, removing the bits it needs from the FIFO in order to do this. It is the encoder's (specifically, the packetiser's) responsibility to ensure that, when the stream is decoded according to this model, the decoder's FIFO neither underflows nor overflows.

Practical decoders will not operate on one sample at a time, but on chunks typically consisting of 64 or 80 samples. In this case, a larger number of bits given by

$$transfer\_unit = data\_rate \times chunk\_size$$

is transferred[25] to the decoder's input and dealt with by the depacketiser before the decoder is asked to decode *chunk_size* samples. In this case the FIFO will not underrun, but maximum occupancy of the FIFO will be greater and the FIFO will need to be larger[26] by one *transfer_unit*.

8.8.2 Demand fed variable-rate decoder

When a variable-rate stream is derived from a fixed-rate stream by omitting the padding (section 8.2), a natural decoding model involves inserting an additional stage before the fixed-rate decoder. The additional stage is a transcoder from variable rate to fixed rate. The transcoder parses the variable-rate stream sufficiently to know where padding is required in the corresponding fixed-rate stream. Data are transferred at a constant rate from the transcoder input to the transcoder output until padding is required, at which point the transcoder input pauses while the output continues at the same rate.

Under this model, the implications for decoder FIFO management are precisely the same as for a fixed-rate stream, as discussed in section 8.8.1. In a practical decoder, the transcoder and depacketiser can be merged into one, and it is not necessary to regenerate and then discard the padding. However, the time taken up by the padding in the notional fixed-rate stream must be accounted for. This ensures that, although the input is demanded at a variable rate, this rate is never greater than that of the notional fixed-rate stream.

8.8.3 MPEG-style decoder

An MPEG model (as used on DVD Audio) can be derived from the above models.

In the MPEG model, unnecessary padding is removed from the access unit or packet, as in the variable-rate model. The external system ensures that each access unit or packet is

---

[25] It is usually convenient to choose *chunk_size* so that it is divisible by an appropriate power of two, so that the *transfer_unit* is an exact number of 16-bit or 32-bit words.

[26] In the case of DVD Audio, the published FIFO size of 90,000 bytes for the standard decoder allows for transfer units equal to one DVD-Audio 'Audio frame', which is 40, 80, or 160 samples at the sampling rates of 48, 96 and 192 kHz respectively. This size assumes that only substream data are transferred to the FIFO; alternative player designs that may pass the whole MLP stream (including the MLP Syncs) to the FIFO will demand a slightly larger FIFO.

delivered to the decoder at a time given by *input_timing*,[27] so that the decoder can synchronise its sample clock with the input just as in the fixed-rate case.

In the MPEG model the complete access unit is considered to be delivered instantaneously to the decoder at the time given by *input_timing*, instead of *input_timing* being the start as in the other models. Thus data will in general arrive sooner under the MPEG model, and the encoder's packetiser must take account of this to ensure that the decoder's FIFO does not overrun.

8.8.4 Peak data rate of MPEG-compliant stream

In practical delivery systems the instantaneous transfer referred to in section 8.8.3 is achieved by buffering, and a definition of data rate is needed. The one adopted is:[28]

$$rate = size[n]/(input\_timing[n+1] - input\_timing[n])$$

where:

rate is the data rate in bits per sample

size[n] is the number of bits in the n^{th} access unit

input_timing is the input timing of the access unit, in samples (the unwrapped values should be used, not the values in the stream which are wrapped to 16 bits).

8.9 Error checking and recovery

Multiple-level error checking is supported by including check words both at the outer (packet or access unit) level and at the substream level. A manufacturer may choose to omit some of the checks in the interests of economy. In particular, as CRC computations on audio data can become expensive, these may be omitted and reliance placed on the simpler parity checks that are also included within the stream. Even though some errors may go undetected in this case, there is provision for a decoder to ensure that these errors cannot produce 'bangs' substantially above the current level of the music.

The error checking, protection and recovery provisions include:

- A CRC on each packet header or major sync
- A CRC *and* a parity check on each segment of audio data carried in a subpacket or access unit
- Within the substream, a CRC on each restart header and block header
- Limits on the maximum gain and the maximum audio output level[29]
- Additional navigation pointers that allow quicker recovery in the event of a Huffman sequence being broken

The encoder can choose to trade off data rate against robustness by omitting some of the less important of the above checks from the bitstream. Conversely, it can choose to insert them more frequently when it is not producing data close to the rate limit.

---

[27] In MPEG terminology, *input_timing* is referred to as DTS, and *output_timing* is referred to as PTS. DVD Audio requires that the *output_timing* of the substreams should be equal.

[28] With DVD Audio the packetiser ensures that the definition of rate satisfies *rate* ≤ 9.6Mbits/sec.

[29] The decoder can cheaply enforce the gain limit at block level; the output limit requires a check on every sample.

9.3 Form A syntax

9.3.1 packet()

packet( )	Encoding	Section
{		
pa	v(16)	9.8.1
pb	v(16)	9.8.1
pc	v(16)	9.8.1
packet_length_lo	u(16)	9.8.2
packet_length_hi	u(8)	9.8.2
channels	u(8)	9.8.3
signature	v(16)	9.8.1
samples_per_packet	v(16)	9.8.4
data_rate	u(16)	9.8.5
sub_packets	v(16)	9.8.5
sub_packet_0_size	u(16)	9.8.5
sub_packet_size	u(16)	9.8.5
sample_number	u(16)	9.8.7
substreams	u(8)	9.8.8
substream_info	v(8)	9.8.9
channel_meaning()		A.1.1
packet_header_CRC	u(16)	9.8.10
sub_packet(sub_packet_0_size)		
`for (s = 1; s < sub_packets; s++)`		
{		
sub_packet(sub_packet_size)		
}		
}		

9.3.2 *sub_packet()*

sub_packet(size)	Encoding	Section
{		
sub_packet_start:		
for (i=0; i < substreams; i++)		
{		
substream_restart[i]	u(16)	9.8.11
substream_start[i+1]	u(16)	9.8.11
}		
for (i=0; i < substreams; i++)		
{		
data_start[i]:		
DATA		9.8.12
substream_parity	u(8)	9.8.13
substream_CRC	u(8)	9.8.13
}		
if (extra_start < sub_packet_start + size)		
{		
extra_start:		
EXTRA_DATA		9.8.12
}		
sub_packet_end:		
}		

9.4 Form B syntax

An *access unit* consists of an *MLP Sync* (from label `mlp_sync` to label `start` in the syntax below) followed by the data for the various substreams. A *major sync* is the same as a *minor sync*, but with additional information contained in the substructure *major_sync_info()*. Major syncs can be distinguished using the fact that the bitfield of 4 bits starting at bit offset 32 from the start of the access unit always equals `0xF`, whereas the minor sync's corresponding bitfield can never take this value.

The *substream_directory* contains end pointers for all the substreams that are represented within the current access unit, and restart pointers for those that have restart headers that are not at the start of their respective DATA regions.

9.4.1 *access_unit()*

access_unit()	Encoding	Section
{		
mlp_sync:		
check_nibble	v(4)	9.9.1
access_unit_length	u(12)	9.9.1
input_timing	u(16)	9.9.1
if (major_sync)		
major_sync_info()		
substream_directory:		
for (i=0; i < substreams; i++)		
{		
restart_pointer_exists	b(1)	9.9.2
restart_nonexistent	b(1)	9.9.2
crc_present[i]	b(1)	
reserved	v(1)	
substream_end_ptr[i]	u(12)	9.9.2
if (restart_pointer_exists)		
{		
reserved	v(4)	
substream_restart[i]	u(12)	9.9.2
}		
else substream_restart[i] = (restart_nonexistent ? null : 0)		
}		
start:		
for (i=0; i < substreams; i++)		
{		
substream_start[i]:		
DATA		9.9.3
if (crc_present[i])		
{		
substream_parity	u(8)	9.8.13
substream_CRC	u(8)	9.8.13
}		
substream_end[i]:		
}		
extra_start:		
EXTRA_DATA		9.9.3
unit_end:		
}		

9.4.2 *major_sync_info()*

major sync info( )	Encoding	Section
{		
format_sync	v(32)	9.9.5
format_info	v(32)	9.9.5
signature	v(16)	9.9.5
flags	v(16)	9.9.6
reserved	v(10)	
channels	u(6)	9.8.3
variable_rate	b(1)	9.9.7
peak_data_rate	u(15)	9.9.7
substreams	u(4)	9.9.8
common_delay_substreams	u(4)	9.9.8
substream_info	v(8)	9.9.8
channel_meaning()		A.1.1
major_sync_info_CRC	u(16)	9.9.10
}		

9.5 Substream syntax

A substream contains a potentially infinite sequence of blocks, punctuated by restart headers. In form A syntax, the restart headers are preceded by padding to a 16-bit boundary; in form B syntax this padding may also precede any block header.

9.5.1 *substream()*

substream()		
{		
while TRUE /* i.e. forever */		
do {		
restart_header()		9.5.2
do		
{		
do		
{		
block()		9.6
}		
while !**pad_to_16**	b(1)	
restart = (syntax==A)|| **next_is_restart**	b(1)	
padding	pad(0 ... 15)	
}		
while !restart		
}		
}		

9.5.2 *restart_header()*

	Encoding	Section
restart_header()		
{		
restart_sync_word	v(16)	9.10.1
min_chan	u(4)	9.10.3
max_chan	u(4)	9.10.3
max_matrix_chan	u(4)	9.10.3
output_timing	u(16)	9.10.4
max_lsbs	u(5)	9.10.5
max_shift	s(4)	9.10.5
max_bits	u(5)	9.10.5
max_bits	u(5)	9.10.5
dither_seed	u(23)	9.10.6
dither_shift	u(4)	9.10.6
error_protect	b(1)	9.10.7
lossless_check	u(8)	9.10.8
reserved	v(16)	
`for (ch=0; ch ≤ max_matrix_chan; ch++)`		
{		
ch_assign[ch]	u(6)	9.10.3
}		
restart_header_CRC	b(8)	9.10.9
}		

After a restart header the decoder's parameters are initialised to default values, which are zero in many cases. The *change* flags (section 9.10.10) are initialised to TRUE.

9.6 Block syntax

A block (see figure 10) consists of audio data, which may be preceded by a block header. Each call to *block_data* will process *block_size* samples. (The `do...while` loop is provided so that real-time encoders with very limited lookahead can set *block_size* to a small value and terminate the block as soon as signal statistics have changed.)

block()		
{		
block_header()		9.6.1
`do`		
{		
`if (error_protect)`		
block_data_bits	u(16)	9.10.7
block_data()		9.6.4
`if (error_protect)`		
block_header_CRC	u(8)	9.10.7
}		
`while` (**extend_block**)	b(1)	
}		

9.6.1 *block_header()*

	Encoding	Section
block_header( )		
{		
if (change_guards && **new_guards**)	b(1)	9.10.10
{ **change_block_size**	b(1)	
change_matrixing	b(1)	
change_output_shift	b(1)	
change_huff_offset	b(1)	
change_coeffs_A	b(1)	
change_coeffs_B	b(1)	
change_quantiser_step_size	b(1)	
change_guards	b(1)	
}		
if (change_block_size && **new_block_size**)	b(1)	
block_size	u(9)	9.10.11
if (change_matrixing && **new_matrixing**)	b(1)	
{ **primitive_matrices**	u(4)	9.10.12
for (i = 0; i < primitive_matrices; i++)		
{		
matrix_ch[i]	u(4)	9.10.12
frac_bits	u(4)	9.10.12
lsb_from_bucket[i]	b(1)	9.10.12
for (j = 0; j ≤ max_matrix_chan+2; j++)		
{		
if (**m_flag**)	b(1)	
m_coeff[i][j]	sfrac(2, frac_bits)	9.10.12
else m_coeff[i][j] = 0		
}		
}		
}		
if (change_output_shift && **new_output_shift**)	b(1)	
{ for (ch = 0; ch ≤ max_matrix_chan; ch++)		
output_shift[ch]	s(4)	9.10.13
}		
if (change_quantiser_step_size		
&& **new_quantiser_step_size**	b(1)	
)		
for (ch = 0; ch ≤ max_chan; ch++)		
{ **quantiser_step_size[ch]**	u(4)	9.10.14
}		
for (ch = min_chan; ch ≤ max_chan; ch++)		
{ if (**params_for_this_chan**)	b(1)	
channel_params(ch)		9.6.2
}		
}		

9.6.2 channel_params()

	Encoding	Section
channel_params(chan)		
{		
if (change_coeffs_A && **new_coeffs[A]**)	b(1)	
{		
new_filter(A, chan)		9.6.3
}		
if (change_coeffs_B && **new_coeffs[B]**)	b(1)	
{		
new_filter(B, chan)		9.6.3
}		
if (change_huff_offset && **new_huff_offset**)	b(1)	
{		
huff_offset[chan]	s(15)	9.10.15
}		
huff_type[chan]	u(2)	9.10.15
huff_lsbs[chan]	u(5)	9.10.15
}		

9.6.3 *new_filter()*

```
new_filter(filter_bank, chan)
 {
   order = order[filter_bank][chan]                                  u(4)    9.10.16

   if (order ≠ 0)
    {
       coeff_Q[chan]                                                 u(4)    9.10.16
       coeff_bits                                                    u(5)    9.10.16
       coeff_shift                                                   u(3)    9.10.16

       for (c = 1; c ≤ order; c++)
        {
           coeff[filter_bank][chan][c]      sshift(coeff_bits, coeff_shift)   9.10.16
        }

       if (new_states)                                               b(1)    9.10.17
        {
           state_bits                                                u(4)    9.10.17
           state_shift                                               u(4)    9.10.17

           for (c = 1; c ≤ order; c++)
            {
               state[filter_bank][chan][c]   s(state_bits, state_shift)   9.10.17
            }
        }
    }
 }
```

9.6.4 *block_data()*

```
                                                                     Encoding   Section
block_data
 {
   for (i = 0; i < block_size; i++)
    {
       for (j = 0; j < primitive_matrices; j++)
        {
           if (lsb_from_bucket[j])
               bucket_lsb[j]                                         b(1)    9.10.18
        }

       for (ch = min_chan; ch ≤ max_chan; ch++)
        {
           audio_data[ch][i]                                         Huff    9.10.19
        }
    }
 }
```

9.7 Alphabetical list of substream variables

This section summarises the encodings and section references for the substream variables. The only stream items omitted are local variables used transiently in decoding the bitstream, and the *change* flags. This list can be used as a basis for allocating the variables in the decoder.

Variable	Encoding	Section

(List of variables, with encoding spec and section number to be inserted)

9.8 Meaning of the form A stream variables

This subsection and the next describe the stream variables that participate in the decoding algorithm. Variables not mentioned here are concerned solely with parsing the stream, and their meaning is implicitly defined by the syntax.

In this subsection we list both the variables that are specific to the form A syntax and those that are common to form A and form B.

9.8.1 IEC sync words P_a, P_b, P_c and signature

The form A packet is intended to be compatible with an IEC958 transport layer. To this end the first four 16-bit words conform to the format of an IEC958 *burst preamble*. The first three of these words are:

P_a = 0xF872

P_b = 0x4E1F

P_c = 0x000E (provisionally)

P_a and P_b are the standard IEC sync words that allow the packet header to be recognised. P_c is the IEC burst_info word. Pending allocation by the IEC of a value to identify an MLP stream, Meridian encoders insert a value of hexadecimal 000E. Current decoders ignore this and inspect the sixth word of the header

signature = 0xB752

for reliable identification of an MLP stream.

9.8.2 packet_length_lo, packet_length_hi

The *packet length* is recorded in every packet even when it is constant, as in a fixed-rate stream. It is measured in bits and expressed as a 24-bit word. This word is divided into the top 8 bits *packet_length_hi* and the bottom 16 bits *packet_length_lo*. These are transmitted in reverse order, i.e. with the 16 least significant bits first.

The packet length is always a multiple of 16 bits, and the items in packets and subpackets are aligned on 16-bit boundaries.

At low data rates the MLP packet length will not exceed 65535 bits, hence *packet_length_lo* describes the length adequately. This word is transmitted in the place in which an IEC 958 layer expects the length code P_d, so that these lower-rate streams are compatible with IEC958.

9.8.3 channels

channels is the total number of channels conveyed in all substreams, including empty channels (see also section 9.10.2). To cater for advanced applications, *channels* can take values in the range 1...63. The standard decoder will not have to decode more than 6 of these.

It is expected that standard DVD decoders will ignore *channels*. Instead, they will determine which substream(s) to decode using *substream_info* (section 9.9.8) and then determine the number of channels from the channel numbers (section 9.10.2) specified in the substream(s).

Since *channels* refers to the whole MLP stream, a particular decoder that will decode a subset should not use *channels* to determine which substreams to decode. Standard decoders should instead use *substream_info* (section 9.9.8) for this purpose.

9.8.4 samples_per_packet

samples_per_packet gives the difference in *input_timing* (section 8.8.1) between the current packet and the next one. For the convenience of the transport layer it may be given a fixed value, such as 1536 sample periods, but this is not an MLP requirement.

9.8.5 data_rate

The first bit of *data_rate* is set to indicate a variable-rate stream, and cleared otherwise.

In a fixed-rate stream, the last 15 bits of *data_rate* specifies the data rate in units of $^1/_{16}$ bit per sample period.

In a variable-rate stream the last 15 bits of *data_rate* specifies the data rate of the corresponding fixed-rate stream

9.8.6 sub_packets, sub_packet_0_size, sub_packet_size

In a form A stream, the packet contains one or more subpackets. The sizes of the subpackets are all multiples of 16 bits and are all the same, with the possible exception of the first. The number of subpackets, the size of the first subpacket (in 16-bit words) and the size of the rest are given by *sub_packets*, *sub_packet_0_size* and *sub_packet_size* respectively.

9.8.7 sample_number

sample_number has the same function in form A syntax as *input_timing* in form B syntax. Both are described in section 8.8.1. *sample_number* is measured in sample periods and is modulo 65536 in the packet header.

9.8.8 substreams

substreams is the number of substreams within the stream, which will be in the range $1 \leq substreams \leq 15$.

9.8.9 substream_info

substream_info is an octet[30] that tells the simpler decoders which substreams they are intended to decode. (More advanced decoders should look at the *extended channel meaning* data (appendix B) for complete information about the contents of the substreams.) In the following, the bit numbers refer to *substream_info*, bit 0 being the last bit, and bit 7 being the first.

2-channel decoder *bit 0* is set if substream 0 should be decoded.

If *bit 0* is clear, a player with a reduced decoder will not be able to decode this bit-stream.

If *bit 0* is set, it follows that substream 0 has at most 2 channels.

Standard decoder *bit 1* is set if substream 0 should be decoded.

bit 2 is set if substream 1 should be decoded.

At least one of these bits will be set. If they are both set, substreams 0 and 1 should both be decoded and the final signal assembled by matrixing the channels from both substreams, as shown in figure 2.

Bits 3–7 of *substream_info* are **reserved.**

[30] An octet is 8 bits.

9.8.10 packet_header_CRC

The *packet_header_CRC* is a 16-bit CRC computed from all the preceding bits generated by the expansion of *packet()*, modulo the polynomial

$$x^{16} + x^5 + x^3 + x^2 + 1$$

(The shift register is cleared before the computation.)

9.8.11 substream_start, substream_restart, data_start, extra_start

substream_start[i] is the starting address of the DATA for substream *i*, relative to the start of the subpacket.[31] i.e.,

substream_start[i] = `data_start[i] - sub_packet_start`

Addresses are assumed to be addresses of 16-bit words.

substream_start[0] does not appear in the bitstream; this is implicitly 2×*substreams*.

substream_start[substreams] is interpreted as the address of the EXTRA_DATA, thus

substream_start[substreams] = `extra_start - sub_packet_start`

size = `sub_packet_end - sub_packet_start`

If *substream_start[substreams+1]* equals *size*, there is no EXTRA_DATA.

substream_restart[i] is the offset (in 16-bit words) of the first restart point (if any) within the DATA for subpacket *i*, i.e.

substream_restart[i] = <*restart point*> `- data_start[i]`

If there is no such restart point, *substream_restart[i]* is set to zero.

9.8.12 DATA and EXTRA_DATA

DATA for each substream consists of a segment taken from the continuous stream of bits representing the substream, whose syntax is given in section 9.5. In a form A stream this segment is broken at an arbitrary 16-bit boundary (see section 8.6).

EXTRA_DATA, if present, allows for extensions to MLP, including *extended channel meaning* (appendix B).

9.8.13 substream_parity and substream_CRC

substream_parity is an 8-bit parity check equal to the exclusive-OR of all the octets in DATA, exclusively-ORed with the constant `0xA9` (the purpose of the latter being to force the check to fail in the event of the stream consisting entirely of zeros).

substream_CRC is an 8-bit CRC computed from all the bits in the preceding DATA modulo the polynomial:

$$x^8 + x^6 + x^5 + x^4 + 1$$

where the relevant shift register is initialised to `0xA2` before the computation.

[31] If *substream_start[i]* = *substream_start[i+1]* then there is no data (nor checkword) for substream i in this subpacket.

9.9 Meaning of the form B stream variables

This section defines the variables that are specific to the form B syntax.

9.9.1 check_nibble, access_unit_length, input_timing

check_nibble is a 4-bit check. In a minor sync, it is calculated so that the exclusive-OR of all the 4-bit nibbles in the minor sync is `0xF`. In a major sync the calculation includes only the items that also appear in a minor sync, i.e. it excludes the expansion of *major_sync_info()* (which is protected by its own CRC).

access_unit_length is the length of the complete access unit, expressed in 16-bit words.

input_timing is the time at which the access unit is passed to the decoder, expressed in sample periods and modulo 65536 (see section 8.8).

9.9.2 substream_restart, substream_end_ptr, restart_pointer_exists, restart_nonexistent

All pointers and offsets within the substream are counting 16-bit words.

restart_pointer_exists is TRUE if there is a restart header that is not at the start of the DATA for the current substream. In this case *substream_restart[i]* is explicitly represented and is the offset of the beginning of the restart header relative to label `substream_start[i]`. If the restart header is at the start of the DATA,[32] the offset is zero, and is not explicitly represented.

restart_nonexistent[33] is TRUE if there is no restart point within the DATA.

substream_end_ptr[i] is the offset of `substream_end[i]` relative to `start`.

9.9.3 DATA and EXTRA_DATA

DATA for each substream consists of a segment taken from the continuous stream of bits representing the substream, whose syntax is given in section 9.5 (see section 8.6).

EXTRA_DATA, if present, allows for extensions to MLP, including *extended channel meaning* (appendix B). It starts at `extra_start`, which is at the same position in the stream as `substream_end[substreams-1]`. Thus, if *substream_end_ptr[substreams-1]* equals `unit_end-start`, there is no EXTRA_DATA.

9.9.4 substream_parity and substream_CRC

See section 9.8.13.

9.9.5 format_sync, format_info and signature

A 32-bit synchronisation word, *format_sync*, is provided close to the start of an MLP major sync so that the major sync can be recognised without additional navigation information.

For a DVD stream, *format_sync* is `0xF8726FBB`. In all cases, the first nibble must be `0xF` so that a major sync can be distinguished from a minor sync.

Further confirmation of a valid major sync is given by the *signature* word:

---

[32] In a DVD Audio stream, the restart header, if it exists, will always be at the start of the DATA.

[33] The reason for the inverted logic is to make it impossible for the first nibble of the substream directory to have the value `0xF`, and thus create ambiguity between major and minor syncs (cf. section 9.9.4).

signature = 0xB752

In a DVD Audio stream, *format_info* consists of the 4 bytes that would be in the private header in an LPCM case. These bytes give the two quantisation wordlengths, the sampling frequencies, the multichannel type and the channel assignment.

9.9.6 flags

Meanings are currently allocated to the first two bits of the *flags* word, the remaining bits being **reserved.** The meaning of a non-zero bit encoding is as follows, where bit 15 is the first bit and bit 0 is the last:

Bit 15 The 'common delay' (section 9.9.8) is constant.

Bit 14 The stream complies[34] with restrictions for DVD Audio that are described externally to this document.

9.9.7 peak_data_rate, variable_rate

In a fixed-rate stream, *peak_data_rate* specifies the data rate in units of $^1/_{16}$ bit per sample period.

In a variable-rate stream, *peak_data_rate* similarly specifies the maximum rate of the stream. In this case the *variable_rate* bit is set.

For use on DVD Audio, the stream is variable rate.

9.9.8 substreams

See section 9.8.8.

9.9.9 common_delay_substreams and substream_info

In some applications, including DVD, the FIFO delay is made equal between substreams. *common_delay_substreams* specifies the number of consecutive substreams (starting from substream 0) that have the same delay. If in addition bit 15 of *flags* (section 9.9.6) is set, this common delay is constant for the whole encoded object.

substream_info is an octet that tells standard decoders which substreams they are intended to decode. (More advanced decoders should look at the *extended channel meaning* data, appendix B, for complete information about the contents of the substreams.) The following applies to MLP streams intended for DVD. The bit numbers refer to *substream_info*, bit 0 being the last bit and bit 7 the first.

2-channel decoder *bit 0* is set if substream 0 should be decoded

bit 1 gives further information (see below).

If *bit 0* is clear, a player with a 2-channel decoder should look elsewhere on the disc for a suitable 2-channel stream.

If *bit 0* is set, it follows that substream 0 has at most 2 channels.

bit 1 is set if a simplified decoder can be used for substream 0. This is used to save decoder MIPS at high sampling rates such as 176·4 and 192kHz. *bit 1* implies that:

[34] Except possibly for differences in packetisation that can be adjusted by a transcoder.

- Re-correlator filter A has order ≤ 4, and filter B does not exist (it has order zero)
- There is no bit-bucket
- In the restart header syntax, *ch_assign[ch]* = *ch*, i.e. there is no re-mapping of the channels

Standard decoder *bit 2* is set if substream 0 should be decoded

bit 3 is set if substream 1 should be decoded.

At least one of these bits will be set. If they are both set, substreams 0 and 1 should both be decoded and the final signal assembled by matrixing the channels from both substreams, as shown in figure 2.

Bits 4–7 of *substream_info* are reserved.

9.9.10 major_sync_info_CRC

The *major_sync_info_CRC* is a 16-bit CRC computed from all the preceding bits generated by the *major_sync_info()* syntax modulo the polynomial

$$x^{16} + x^5 + x^3 + x^2 + 1$$

(The shift register is cleared before the computation.)

9.10 Substream syntax variables

The substream syntax is common to the form A syntax and the form B syntax.

9.10.1 restart_sync_word

restart_sync_word has the value `0xF1EA`. It can be used to confirm the existence of a restart header.

9.10.2 Channel numbers

There are two types of channel number, local and global. Channels are referred to locally within the substream (or within a group of related substreams) by local channel number. Within the substreams intended for decoding by a standard DVD decoder, the local channel numbers are restricted to the range 0 to 5. If substream 0 is intended to be decoded by a 2-channel decoder, then:

$$0 = min\_chan \le max\_chan \le 1$$

9.10.3 min_chan, max_chan, max_matrix_chan, ch_assign

min_chan and *max_chan* are local channel numbers giving the minimum and the maximum channel number carried by the substream. If *min_chan* is zero, the substream can be decoded independently. If *min_chan* is non-zero, partially decoded channels from previous substreams must be assembled in order to feed the final matrixing, as in figure 2.

The matrixing may result in more channels than were input to it. Its output channels are numbered from 0 to *max_matrix_chan*.

ch_assign[ch] is the global channel number referring to the outputs of the complete decoder. The decoded channel with local channel number *ch* is routed to the output with channel number *ch_assign[ch]*. In substreams intended to be decoded by a standard DVD decoder, *ch_assign[ch]* will not exceed 5. In more general contexts it lies in the range 0...*channels*.

With the exception of *ch_assign[ch]*, all channel numbers appearing within the substream syntax are local channel numbers.

9.10.4 output_timing

Like *input_timing* (section 9.9.1), *output_timing* is recorded modulo 65536. It is the sample number of the first sample in the first block after the restart header. The use of *input_timing* and *output_timing* to control the timing of the decoder is described in section 8.8.

9.10.5 max_lsbs, max_shift and max_bits

max_lsbs, max_shift and *max_bits* are all safety features designed to prevent bit errors from producing clicks or bangs substantially louder than the current level of the music. They may be used as well as or instead of the parity, CRC and other error detection mechanisms discussed in sections 9.8.12 and 9.10.8.

max_lsbs and *max_shift* are the maximum values of *huff_lsbs* and *output_shift* specified in any block header up to the next restart point. These checks are extremely cheap, as they can be enforced at block level.

max_bits is the maximum number of bits exercised for any sample and any channel in this substream,[35] up to the next restart point. It takes the value 24 with peak-level signals, and is included twice within the restart header to provide security against corruption of the header itself.

9.10.6 dither_seed and dither_shift

The lossless decoder includes a dither generator generating two independent 8-bit dither channels. The generator uses a shift register sequence of length $2^{23}-1$ from the polynomial

$$x^{23} + x^5 + 1$$

and the shift register is initialised to the value *dither_seed* at each restart point in order to synchronise it with a matching dither generator in the encoder.

On each sample, the shift register is shifted by 16 bits and the result is split into two 8-bit bytes. These bytes are sign extended and shifted by *dither_shift* bits to provide two phantom dither channels (the first 8 bits go to channel *max_matrix_chan*+1, the second 8 bits to *max_matrix_chan*+2) which participate in the matrixing. By providing appropriate matrix coefficients, these dither signals can be added or subtracted, thus providing two independent TPDF dither signals.

dither_shift specifies a left-shift in the range $0 \le dither\_shift \le 15$. When *dither_shift* is 0, the 8-bit dither is right justified within the 24-bit word assumed for signals (i.e. the dither is correctly aligned for truncation to 24 bits when multiplied by a matrix coefficient of 2^{-8}).

9.10.7 error_protect, block_data_bits, block_header_CRC

The encoder may optionally include additional error protection within the substream. If the *error_protect* flag is set, the additional words *block_data_bits* and *block_header_CRC* are present.

block_data_bits is the number of bits to be read by the following call of *block_data()*. This provides a navigation aid so that players can restart at the next block if bit errors within the Huffman-encoded data cause the wrong number of bits to be swallowed.

block_header_CRC provides an additional check on correct navigation, as well as checking for data errors in the block header. It is a CRC on the block header, using the same generating polynomial for *restart_header_CRC* (section 9.10.9). Its shift register is initialised to `0xA2`.

[35] Including the channels produced by matrixing earlier substreams.

9.10.8 lossless_check

To allow the decoder to verify that the reconstruction is indeed lossless, a *lossless_check* octet is included in each restart header, consisting of an exclusive-OR of the octets in all the decoded samples since the *previous* restart point. All samples in channels 0 ... *max_matrix_chan* are considered, and samples on channel *i* are rotated left by *i* bits before being processed.[36]

9.10.9 restart_header_CRC

restart_header_CRC is a CRC calculated on the restart header up to (but not including) the *restart_header_CRC* itself. The shift register is initialised to zero and the polynomial used is:

$$x^8 + x^4 + x^3 + x^2 + 1$$

9.10.10 'change' and 'new' flags

After a restart header, the decoder's parameters are initialised to default values, which are zero in many cases. Each block header has the opportunity to re-specify some of the parameters. In order to economise on block overheads, each parameter or group of parameters has associated with it a *new* flag which is set if that parameter (or group) is to be encoded within the block header. Thus, parameters that have not changed can be represented by a single bit.

However, the total number of possible *new* flags is over 20 for a 2-channel signal. This would add significantly to the data rate if the encoder were to choose short block lengths in order to change a certain parameter rapidly on transient material.

The encoder therefore also has at its disposal a set of *change* flags. These are initialised to TRUE at a restart header, and can subsequently be reset to FALSE to signify that the corresponding parameter is not subject to rapid change. The *new* flag for a parameter is not transmitted if its *change* flag is FALSE, and block overheads are thus minimised. If *new_guards* is set, the *change* flags can themselves be changed.

9.10.11 block_size

block_size specifies the number of audio samples per channel encoded in a block. The maximum block size is 511 samples.

9.10.12 primitive_matrices, m_coeff, frac_bits, matrix_ch, lsb_from_bucket

primitive_matrices specifies the number of primitive matrices used in the lossless matrixing. On DVD, *primitive_matrices* will not exceed 6 in the substreams that are intended for decoding by a standard player, or 2 in a substream 0 that is intended for a 2-channel decoder.

The i^{th} primitive matrix modifies channel *matrix_ch[i]*, which lies in the range $0 \le$ *matrix_ch[i]* $\le$ *max_matrix_chan*. *m_coeff[i][j]* specifies the proportion of channel *j* that contributes to the new *matrix_ch[i]*, where $0 \le j \le$ *max_matrix_chan*+2 and thus includes the dither channels (section 9.10.6).

[36] Each 24-bit sample is considered as being expanded as 3 bytes for the purpose of this check. However, on a 24-bit processor it will probably be more convenient to accumulate a 24-bit-wide exclusive-OR of the decoded samples and collapse this to 8 bits afterwards. It does not matter whether the rotation is performed on the 24-bit value or the collapsed 8-bit value.

frac_bits, in the range $0 \le frac\_bits \le 14$, is the number of bits after the binary point in the encoded representation of *m_coeff[i][j]*. The coefficient range is $-2 \le m\_coeff[i][j] < 2$, and the extreme value of –2 will be exercised.

The flag *lsb_from_bucket[i]* states whether the encoder's primitive matrix has dropped an lsb into the bit-bucket (section 7.3), so that the decoder's i^{th} primitive matrix needs to retrieve it.

9.10.13 output_shift

After matrixing, channel *ch* is shifted by *output_shift[ch]* bits. A positive value indicates a left-shift, and the range is $-8 \le output\_shift \le +7$.

The comparison of signal levels with *max_bits* (section 9.10.4) is to be made after this shift.

9.10.14 quantiser_step_size

quantiser_step_size[ch] specifies the step size of the quantisers used in the processing of channel *ch*, where $0 \le ch \le max\_chan$. This step size is used in the Huffman coding, in the re-correlator and also in any primitive matrices that modify channel *ch*.[37]

The step size is expressed as a left-shift relative to the lsb of the 24-bit input signal, i.e. an encoded value of 3 corresponds to a step size of 8 lsbs.

9.10.15 huff_type, huff_lsbs, huff_offset

huff_type[ch] selects one of four Huffman decoding strategies optimised for assumed signal distributions as follows:

0	Rectangular
1	Rectangular with exponential tails
2	Laplacian (Rice code)
3	Laplacian with narrower central peak

Huffman coding is applied to the higher-order bits of the signal word, the remainder being transmitted verbatim. *huff_lsbs[ch]* specifies how many lsbs of the signal word are to be transmitted verbatim.

The encoder determines *quantiser_step_size[ch]* by examining the unused lsbs in the input signal on each block,[38] such that *quantiser_step_size[ch]* lsbs are known to be zero and are therefore not transmitted.

The assumed signal distributions are symmetrical about zero, so *huff_offset[ch]* is added to allow for asymmetrical signals, DC components and low-frequency components that may appear DC-like on a timescale of one block.

The sample value given by *audio_data[ch][i]* is retrieved and decoded as follows:

1. Determine the msbs by reading a variable number of bits from the substream, and decode according to the distribution specified by *huff_type[ch]*.
2. Shift left by *huff_lsbs[ch]–quantiser_step_size[ch]* bits.
3. Read *huff_lsbs[ch]–quantiser_step_size[ch]* bits from the substream into the vacated lsbs.
4. Add *huff_offset[ch]*.
5. Shift left by *quantiser_step_size[ch]* bits, shifting zeros into the lsbs.

---

[37] If a primitive matrix modifies a channel in the range $max\_chan < ch \le max\_matrix\_chan$, a step size of unity is assumed.

[38] The input shift is also taken into account (section 7.1).

The number of transmitted lsbs (*huff_lsbs[ch]–quantiser_step_size[ch]*) may well be zero, and this is used with a rectangular distribution to achieve an efficient coding of digital black.

9.10.16 coeff, order, coeff_Q, coeff_shift

The orders of filters A and B in figure 8b are specified by *order[A][ch]* and *order[B][ch]* respectively, where

$$order[A][ch] \leq 8$$

$$order[B][ch] \leq 4$$

$$order[A][ch] + order[B][ch] \leq 8$$

The coefficients are given by *coeff[A][ch][c]* and *coeff[B][ch][c]*. Conceptually these coefficients are fractional, but to simplify implementation on a fixed-point processor, the model used is of 16-bit integer coefficients followed by a right-shift of *coeff_Q*, where $8 \leq coeff\_Q \leq 15$. It is at the implementor's discretion whether the output of the filter is shifted right, or whether the filter coefficients are shifted when they are read in. To facilitate the decoder design:

- the encoder will always specify the same *coeff_Q* value for the filters A and B.
- if *coeff_Q* changes, the coefficients of *both* the filters A and B will be re-specified, except that if either is the trivial filter (zero coefficients) it need not be re-specified.

The 16-bit coefficients are specified by reading a signed integer with *coeff_bits* bits and shifting left by *coeff_shift*. These values obey the constraints

$$1 \leq coeff\_bits \leq 16$$

$$0 \leq coeff\_shift \leq 7$$

$$coeff\_bits + coeff\_shift \leq 16$$

These 16-bit integer coefficients will never have the extreme negative value of –32768.

9.10.17 state

The states (delayed variables) of the filters A and B are initialised to zero at a restart point.

Subsequently, the encoder has the option to set the state of the filter B[39]. *state[B][ch][n]* is the value of the n^{th} delayed variable in filter B. It is encoded as a signed integer[40] with *state_bits* bits, which is left-shifted by *state_shift* bits before being loaded into the delayed variable.

9.10.18 bucket_lsb

If the bit-bucket has been used (*lsb_from_bucket[j]* = TRUE, section 9.6.4), *bucket_lsb[j]* is the discarded bit to be inserted in the j^{th} primitive matrix in the decoder. See also section 7.3.

9.10.19 audio_data

audio_data[ch][i] is the audio data for the block, Huffman coded using the parameters discussed in section 9.10.15, with the channels interleaved.

[39] The syntax allows the states of both filter A and filter B to be set. However, it is not envisaged that encoders will set the state of filter A, an action which is in any case prohibited on DVD Audio.

[40] Internally, signals are considered as 24-bit integers within the assumed 24-bit data path.

10 References and bibliography

[1] Craven, P.G. and Gerzon, M.A., 'Lossless Coding for Audio Discs', *J. Audio Eng. Soc.*, vol. 44 no. 9 pp. 706–720 (September 1996)

[2] Craven, P.G., Law, M.J. and Stuart, J.R., 'Lossless Compression using IIR Prediction Filters', *J. Audio Eng. Soc.* (Abstracts), vol. 45 no. 5 p. 404 (March 1997) (Preprint #4415)

[3] Stoll, G., Nielsen, S. and van der Kerkhof, L., 'Generic Architecture of the ISO/MPEG Audio Layer I and II: Compatible Developments to Improve the Quality and Addition of New Features'. Preprint 3697 of the 95th AES Convention, New York (October 1993)

[4] Craven, P.G. and Gerzon, M.A., 'Lossless Coding Method for Waveform Data', International Patent Application no. PCT/GB96/01164 (May 1996)

[5] Craven, P.G. and Law, M.J., 'Reference Decoder for the MLP Bitstream', Meridian Audio Ltd

[6] *Acoustic Renaissance for Audio*, 'A Proposal for High-Quality Application of High-Density CD Carriers', private publication (April 1995)

[7] Craven, P.G. and Stuart, J.R., 'Cascadable Lossy Data Compression Using a Lossless Kernel', *J. Audio Eng. Soc.* (Abstracts), vol. 45 no. 5 p. 404 (March 1997) (Preprint #4416)

[8] Craven, P.G. and Gerzon, M.A., 'Compatible Improvement of 16-Bit Systems Using Subtractive Dither', *AES 93rd Convention*, San Francisco, preprint 3356 (1992)

[9] Gerzon, M.A., Craven, P.G., Stuart, J.R. and Wilson, R.J., 'Psychoacoustic Noise Shaped Improvements in CD and Other Linear Digital Media', *AES 94th Convention*, Berlin, preprint 3501 (March 1993)

[10] Stuart, J.R. and Wilson, R.J., 'Dynamic Range Enhancement Using Noise-shaped Dither Applied to Signals with and without Pre-emphasis', *AES 96th Convention*, Amsterdam, preprint 3871 (1994)

[11] Stuart, J.R. and Wilson, R.J., 'Dynamic Range Enhancement using Noise-Shaped Dither at 44.1, 48 and 96 kHz', *AES 100th Convention*, Copenhagen (1996)

Appendices

A. Channel meaning information

As well as delivering audio losslessly, MLP provides additional *channel meaning* data that describe the audio. This appendix summarises this information without giving specific details. These details (for example, bit encodings) will be described in supplements to this document.

The *channel meaning* data are embedded in the external interface, i.e. in the packet (section 9.3.1) or the major sync (section 9.4.2). The information is thus refreshed quite frequently.[41] Its amount has been limited to 64 bits to avoid increasing the data rate significantly. More comprehensive information can be found in the *extended channel meaning* data (appendix B).

A.1.1. *channel_meaning()*

channel_meaning() {	Encoding	Section
fs	u(5)	A.2.1
wordwidth	u(5)	A.2.2
channel_occupancy	v(6)	A.2.3
multi_channel_type	u(3)	A.2.4
speaker_layout	u(10)	A.2.6
copy_protection A.2.7		u(3)
level_control	b(16)	A.2.8
reserved	v(7)	
source_format	u(4)	A.2.5
summary_info	b(5)	A.2.9
}		

A.2. Channel meaning variables

A.2.1. fs

fs (sampling frequency) is an enumeration of standard sampling frequencies in the range 8kHz to 384kHz.

A.2.2. wordwidth

wordwidth is an integer in the range 0–24. If *wordwidth* is less than 24, then (24–*wordwidth*) least significant bits are guaranteed to be zero (on all channels) throughout the current audio object.

A.2.3. channel_occupancy

channel_occupancy is a 6-bit mask that registers whether or not channels 0–5 are occupied (channel 5 is first and channel 0 last). This applies to an audio object as a whole: the mask should not be set to 0 just because a particular channel is inactive for a period of time.

[41] Typically every 7mS with DVD Audio.

A.2.4. multi_channel_type

multi_channel_type describes the following possibilities:

- Standard speaker layout
- Standard speaker layout with height
- Non-standard speaker layout
- Not speaker feeds

A.2.5. source_format

source_format indicates the multichannel system being used. The types described include

- Unclassified
- MSTBFZ (hierarchical)
- WXYZUV (Ambisonic)
- WXYZEF (Ambisonic)
- Dolby surround (MP matrix)
- UHJ
- Binaural

A.2.6. speaker_layout

speaker_layout is used to describe a variety of horizontal and 3-D speaker layouts, including horizontal and vertical scaling for preferred aspect ratios.

A.2.7. copy_protection

copy_protection includes the simple categories 'unrestricted', 'copy once' and 'don't copy'.

A.2.8. level_control

level_control is 16 bits of control information allowing the level and dynamics to be controlled. The MLP decoder does not interpret these bits, but exports them for subsequent processing by a player. The format of these bits is thus dependent on the application. For example:

- 6 bits could be allocated to indicate absolute SPL, and 10 bits for dynamic compression[42]
- 8 bits could be allocated to each of two compression signals. If the $\{L_0, R_0\}$ feature is used, separate compression signals can be carried for the 2-channel and multichannel mixes.

A.2.9. summary_info

summary_info provides a compact encoding of common situations described in more detail by *multi_channel_type*, *source_format* and *speaker_layout* (such as 5·1, 5·0, 5·0 with height, 2+2 Ambisonic). Simple players can use a table-driven interpretation of the *summary_info* bits instead of extracting more comprehensive information from these three fields.

[42] The compression signal should be slowly varying, as its timing relationship to the audio is not precisely defined.

B. Extended channel meaning

Extended channel meaning information is carried in the EXTRA_DATA of the subpacket (section 9.3.2) or access unit (section 9.4.1). The *extended channel meaning* is refreshed by means of selective updates, so that a large number of slowly changing data can be conveyed with minimal impact on data rate. Supplements to this document will describe the format of the EXTRA_DATA field and the *extended channel meaning*.

First-generation decoders will not decode the *extended channel meaning*.

C. SMPTE Time code

Time-code updates can be carried in the EXTRA_DATA, in a manner to be described in a supplement to this document.

The format is provisionally four bytes to carry the time code in the standard *hh:mm:ss:ff* format, followed by a 16-bit *sample_number*. The time-code update is carried out when the sample clock in the decoder (cf. sections 8.8 and 8.8.1) equals the *sample_number* in the time-code update, so that synchronisation is exact to one sample.

The update frequency is at the discretion of the encoder. It is envisaged that receiving equipment will calculate the time code at intermediate points by using the sample clock, so that time-code updates need only be recorded infrequently.